VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Benchmarking prostředí pro rychlý vývoj mobilních her

Benchmarking of Multi-platform Mobile Game Development

2020 Milan Hlavsa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Milan Hlavsa**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Benchmarking prostředí pro rychlý vývoj mobilních her
Benchmarking of Multi-platform Mobile Game Development

Jazyk vypracování: čeština

Zásady pro vypracování:

V současnosti existuje celá řada frameworků (aplikačních rámců) a platforem, které mají zajišťovat rychlý vývoj 2D nebo 3D her v prostředí mobilní platformy. Cílem práce je tvorba sady testovacích příkladů pro vyhodnocení efektivity tří nebo více herních frameworků resp. platforem (např. Phaser3, pixi.js, EASELJS pro 2D hry, resp. BabylonJS, ImpactJS a Unity pro 3D hry).

1. Prostudujte, jaké jsou nejčastěji používané frameworky resp. celé platformy pro vývoj (multiplatformních) mobilních her.
2. Popište, jaké výhody (a nevýhody) mají herní frameworky vůči přímému použití OpenGL/WebGL resp. plátna a srovnejte je mezi sebou.
3. Navrhněte sadu testů, které umožní otestovat výkon jednotlivých částí herního engine (2D nebo 3D grafika, animace, zvuky, detekce kolizí, apod.).
4. Navržené testy implementujte ve zvolených implementačních prostředích.
5. Testy spusťte alespoň na 2 různých mobilních zařízeních s odlišnými verzemi OS popř. i v desktopovém prostředí, a získaná data vyhodnoťte. Získané výsledky popište a zhodnoťte, kdy je použití kterého engine vhodné.

Seznam doporučené odborné literatury:

[1] HTML5 Game Engines Overview. [online] [cit. 2018-09-18] Dostupné z: http://html5gameengine.com/
[2] Dmitry Volevodz: iOS 7 Game Development. Packt Publishing Ltd., 2014. ISBN 978-1-78355-157-6.
[3] Meier, R.: Professional Android 4 Application Development, Wrox, 2012, ISBN-13: 978-1118102275.
[4] Phaser: A fast, fun and free open source HTML5 game framework [online] [cit. 2018-09-18] Dostupné z: http://phaser.io/
[5] Babylon.js Documentation [online] [cit. 2018-09-18] Dostupné z: https://doc.babylonjs.com/
[6] Unity [online] [cit. 2019-05-13] Dostupné z: https://unity.com/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Moravec, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě :9.5.2020…… ……

podpis studenta

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce, panu Ing. Pavlu Moravcovi Ph.D. za velice užitečné konzultace a rady po celou dobu práce. Dále bych rád poděkoval panu Vladimíru Bednářovi a Danielu Zbořilovi za pomoc při testování a hledání nedostatků.

Abstrakt

Práce se zabývá testováním různých vývojových prostředí nebo knihoven pro vývoj multiplatformních her. Je navrženo a implementováno několik testů pro zjištění a porovnání výkonu mezi vybranými prostředími, respektive frameworky. Vybranými frameworky jsou Phaser, BabylonJS a Pixi.js. Všechny testy byly implementovány v programovacím jazyce JavaScript. Testy také zjišťují náročnost na výkon za použití často využívaných funkcí při vývoji her. Na konci jsou výsledky vyhodnoceny.

Klíčová slova

Benchmark, framework, snímky za sekundu, FPS, Phaser, Pixi.js, BabylonJS

Abstract

This thesis deals with testing of the performace of various development enviroments or frameworks for the development of multiplatform games. Several tests are designed and implemented to determine and compare performance between selected enviroments or frameworks. Selected frameworks are Phaser, BabylonJS and Pixi.js. All tests were implemented in the JavaScript programming language. Tests also determine witch frequently used features in game development are more performance intensive than others. At the end, the results are evaluated.

Keywords

Benchmark, framework, frames per second, FPS, Phaser, Pixi.js, BabylonJS

Obsah

Seznam ilustrací a tabulek…….....9

1 Úvod……..…….……10

2 Nejčastěji používané prostředí pro vývoj multiplatformních her…….…….....11

2.1 GDevelop…….11

2.2 BabylonJS……11

2.3 ImpactJS Engine……..…12

2.4 EaselJS…….....12

2.5 Phaser 3……12

2.6 Unity……13

2.7 Pixi.js……..….13

3 Výhody a nevýhody použití frameworků……...……14

3.1 Použití frameworků vůči přímému použití WebGL/Canvas API……14

4 Způsob měření a návrhy testů…….…15

4.1 Měření výkonu…….……....15

4.2 Navržené univerzální testy……..15

4.2.1 Test s kryptografickou hashovací funkcí……...……16

4.2.2 Test se zvukovými formáty……..16

4.2.3 Test s vytvářením 2D objektů……..…….….17

4.3 Navržené testy v 3D prostředích..…….…. 17

4.3.1 Test s vytvářením 3D objektů……...18

4.3.2 Test se světly a vykreslováním stínů……...….19

4.3.3 Test – padání kuliček……...…….19

5 Implementace testu s vytvářením 3D objektů…….……..….21

5.1 Implementace ...……...……...….21

5.2 Implementace testu s kryptografickou hashovací funkcí…….……21

5.3 Implementace testu se zvukovými formáty…….....22

5.4 Implementace testu s vytvářením 2D objektů a konfigurační scény...…….……..…….22

5.4.1 Implementace konfigurační scény……22

5.4.2 Implementace testu s vytvářením 2D objektů……..24

5.5 Implementace testu s vytvářením 3D objektů……..27

5.6 Implementace testu se světly a vykreslováním stínů……...….29

5.7 Implementace testu – padající kuličky……...…….31

5.8 Export a implementace ukládání souboru……...….31

6 Testování a výsledky……..33

6.1 Výsledek testu s kryptografickou hashovací funkcí…….....34

6.2 Výsledek testu se zvukovými formáty…….……34

6.3 Výsledek testu s vytvářením 2D objektů…….……35

6.4 Výsledek testu s vytvářením 3D objektů…….....36

6.5 Výsledek testu se světly a vykreslováním stínů……..37

6.6 Výsledek testu – padající kuličky……38

7 Závěr……...39

Literatura……...40

Seznam příloh……42

Seznam ilustrací a tabulek

Obrázek 1 - Ukázka průběhu testu s kryptografickou hashovací funkcí (Pixi.js)……..……..16

Obrázek 2 - Ukázka průběhu testu s vytvářením 2D objektů (Phaser)……..17

Obrázek 3 - Ukázka průběhu testu s vytvářením 3D objektů (BabylonJS)……..……..…...18

Obrázek 4 - Ukázka průběhu testu se světly a vykreslováním stínů (BabylonJS)……..…….19

Obrázek 5 - Ukázka průběhu testu – padání kuliček (BabylonJS)……..……..…19

Obrázek 6 - Scéna s možností konfigurace testu s vytvářením 2D objektů (BabylonJS)…….22

Obrázek 7 - Scéna s možností konfigurace testu s vytvářením 2D objektů (Pixi.js)…….23

Obrázek 8 - Scéna s možností konfigurace testu s vytvářením 2D objektů (Phaser)……23

Obrázek 9 - Sprite sheet vytvářeného 2D objektu…….24

Obrázek 10 - Typy objektů, které je možné použít v testu s vytvářením 3D objektů (BabylonJS)..27

Obrázek 11 - Graf výsledků testu s k. hashovací funkcí……...34

Obrázek 12 - Grafy výsledků testu s vytvářením 2D objektů - 1. konfigurace…….35

Obrázek 13 - Grafy výsledků testu s vytvářením 2D objektů - 2. konfigurace…….35

Obrázek 14 - Grafy výsledků testu s vytvářením 3D objektů - 1. konfigurace…….36

Obrázek 15 - Grafy výsledků testu s vytvářením 3D objektů - 2. konfigurace…….36

Obrázek 16 - Graf výsledků testu se světly a vykreslováním stínů……...37

Obrázek 17 - Graf výsledku testu – padající kuličky……..…….38

Tabulka 1 - Specifikace použitých zařízení……..33

Tabulka 2 - Přehled podporovaných/nepodporovaných zvukových formátů……...34

1 Úvod

Cílem práce je otestovat výkon frameworků (knihoven) nebo celých vývojových prostředí pro vývoj multiplatformních her. Mobilní hry se stávají čím dál více populární. Dokonce slavné počítačové hry pro více hráčů je v dnešní době možné hrát i na mobilních zařízeních. Známým příkladem může být hra s názvem Fortnite, která si v letech 2017 a 2018 získala popularitu a obrovské množství hráčů, a proto se dočkala i portu na mobilní zařízení. Jelikož se jedná o původně počítačovou hru, lze očekávat, že bude v mobilní verzi velmi výkonnostně náročná, a to i přes jakékoliv grafické změny, které by mohli náročnosti na výkon ulehčit. Nejen z tohoto důvodu je dobré vědět v jakých prostředích nebo frameworcích je vhodné vyvíjet výkonnostně náročnější hry a také co dostupné frameworky nabízí za funkce.

Nejdříve bylo potřeba zjistit, která jsou nejčastěji používaná (prostředí), jaké funkce nabízí a zda slouží k vývoji her v 2D nebo 3D grafice, popřípadě obojího. Také byly prozkoumány výhody a nevýhody použití frameworků. Poté byla zvolena měřící jednota, která se bude používat pro měření výkonu testů. Jedná se o snímky za sekundu, které se dnes běžně používají v herním průmyslu jako jednotka pro plynulost obrazu.

Samotná výkonnostní náročnost se dá testovat různými způsoby. Nejjednodušším způsobem je spustit složitou výpočetní funkci. Tato metoda ovšem není ideální, protože neotestuje výkon jednotlivých frameworků nýbrž výkon samotného zařízení, na kterém je test spuštěn. Ideálnějším způsobem testu je vykreslování různých objektů na plátno, protože v každém frameworku je vykreslování řešeno jinak.

Následně byly testy navrženy. Některé z nich lze provádět pouze v 3D grafice, zatímco jiné lze otestovat všude. Testy byly navrženy tak, aby vyzkoušely zátěž výkonu s použitím jednotlivých funkcí, které jsou při vývoji her využívány. Jedná se například o animace, pohyb nebo kolizi objektů. V testech se nachází spousta náhodných prvků, proto je výsledek testů reprezentován průměrnou hodnotou snímků za sekundu. Poté byly testy implementovány s použitím zvolených frameworků, konkrétně Phaser a Pixi.js pro 2D grafiku a BabylonJS pro 3D. Celá implementace testů proběhla v programovacím jazyce JavaScript.

Následovalo samotné testování, které se uskutečnilo na dvou zařízeních. První bylo mobilní zařízení s operačním systémem Android. Druhým zařízením byl tablet s operačním systémem Windows, avšak testy zde probíhaly ve webovém prostředí, konkrétně prohlížeč Mozilla Firefox. Export na mobilní zařízení byl vyřešen pomocí Apache Cordova. Z důvodu náhodných prvků v testech byl každý test spuštěn vícekrát a ze získaných hodnot se vypočítala finální průměrná hodnota. Tyto finální hodnoty byly pro lepší vizualizaci vloženy do grafů. Výsledky jednotlivých testů byly zhodnoceny a porovnány.

2 Nejčastěji používané prostředí pro vývoj multiplatformních her

Nejdříve je potřeba si objasnit, co je to herní engine a framework. Herní engine je vývojové prostředí obvykle s grafickým rozhraním. Dále tyto enginy nabízí obrovskou škálu funkcí a možností, což značně zjednodušuje vývoj her. Díky grafickému rozhraní je možné vytvořit scénu bez psaní kódu. Tvorba může probíhat pouze přetahováním objektů do scény a přizpůsobováním jejich parametrů, než vznikne herní úroveň. Chování hry je už obvykle potřeba naprogramovat, avšak nabízené funkce a možnosti těchto enginů implementaci značně usnadňují. Například není třeba řešit implementaci simulace herní fyziky, stačí pouze nastavit, na které herní objekty se fyzika vztahuje a po startu aplikace ji engine začne pro vybrané objekty simulovat.

Frameworky pro vývoj her stejně tak jako herní enginy zjednodušují práci a nabízí spoustu funkcí a možností. Na rozdíl od herních enginů frameworky obvykle nemají grafické rozhraní a celá tvorba probíhá psaním kódu, naprogramováním a podobně. Také z důvodu absence grafického prostředí probíhá vykreslování objektů v HTML5 elementu canvas neboli plátno. Pro vykreslování do plátna využívá WebGL API nebo Canvas API.

2.1 GDevelop

Jedná se o open-source herní engine s grafickým prostředím, ve kterém lze vyvíjet 2D hry bez programovacích znalostí. Chování hry lze totiž řídit pomocí tzv. událostí (eventů), jak je zmíněno v [2]. Příklad události: Pokud je herní objekt hráč v kolizi s objektem nepřítel, hráč je zničen, a to celé bez napsání jediného řádku kódu. Toto je velká výhoda pro uživatele, kteří nejsou tolik zdatní v programování. Pokud je to potřeba, je možné vytvořit skripty v jazyce JavaScript, které dokážou spolupracovat s událostmi a slouží k složitějším operacím, které samotné události nezvládnou. Dále nabízí využití herních vlastností jako kolize, simulace herní fyziky, zvuky a další. Dle [1], GDevelop momentálně používá pro vykreslování grafiky jiný framework, a to Pixi.js. Mezi jeho další výhody patří jednoduchý export do mnoha platforem (iOS, Windows, Linux, Mac a další). Jako jeden z mála frameworků na trhu obsahuje export přímo do Facebook Messenger Instant Games. Také obsahuje různé předpřipravené typy her (například skákání přes plošiny nebo střílení z vesmírné lodi), ze kterých lze vycházet při tvorbě vlastního projektu.

Při tvorbě složitějších her nemusí být tento framework vhodný, což je zároveň i jeho nevýhodou. Vývoj v plném grafickém prostředí může být sice jednoduchý, ale při složitějších operacích může být omezující. Kvůli tomu se tato jednoduchost může změnit na komplikované řešení kódem, které zároveň musí pracovat s událostmi. Podle [3] je také pomalý, což dokáže prodloužit dobu vývoje.

2.2 BabylonJS

Open-source framework pro vývoj 3D projektů, respektive her, který je založen na WebGL (JavaScript API pro vykreslování výkonné 2D a 3D grafiky bez použití pluginů, viz [4]). Díky tomu je schopný vykreslovat objekty v realistické podobě. Jelikož se jedná o 3D, tento framework dále nabízí kamery, světla a vykreslování stínů. BabylonJS nemá sám o sobě dostupnou herní fyziku, avšak velice dobře spolupracuje s JavaScriptovými knihovnami, které tuto funkci nabízí (viz [6]). Je možné si vybrat ze čtyř, a to jsou Cannon.js, Oimo.js, Energy.js a Ammo.js. Výhodou tohoto frameworku je také rozsáhlá dokumentace, velká komunita a mnoho tutoriálů. Důležité je zmínit

existenci tzv. BabylonJS-Playground (dostupné na [5]), kde si uživatelé mohou vyzkoušet své kódy přímo na webových stránkách tohoto frameworku a vidět výsledek v reálném čase. Výsledná práce v BabylonJS-Playground může být dostupná i pro jiné uživatele, a proto je jednoduché prezentovat možný problém nebo řešení.

BabylonJS je víceméně založený na psaní kódu, tedy na znalosti programování. To znamená, že nemá grafické prostředí (jako například Unity3D nebo GDevelop). Tento framework byl vybrán pro testování.

2.3 ImpactJS Engine

Jde o open-source framework pro vývoj 2D her. Tento framework se liší od ostatních vlastním zabudovaným editorem herních map (levelů), který funguje na principu mřížky, proto je vhodný zejména pro vývoj plošinových her, nebo starších herních hitů (Tetris, Pong…). Dle [7] nabízí dva druhy kolizí, statické a dynamické.

Nevýhodou tohoto frameworku je, že není příliš udržovaný, jeho poslední aktualizace byla v roce 2014. Pravděpodobně kvůli tomu toho opravdu mnoho nenabízí oproti ostatním frameworkům. Téměř veškerá práce probíhá se třídou Entity (viz [8]), která reprezentuje herní objekt a jako jediná nabízí lepší možnosti jako animace, kolize a podobně. Ostatní třídy nemají tolik možností, ku příkladu třída Font (viz [9]) nenabízí dynamickou změnu textu, jakmile je text jednou vytvořen, není možné ho změnit, což může například zkomplikovat tvorbu herního menu. Podle mého individuálního názoru, pokud uživatel nevyužívá zabudovaný generátor map, je pro něj opravdu náročné něco vytvořit a pro tvorbu složitějších 2D herních titulů není tento framework vhodný vůbec.

Tento framework byl vybrán pro testování, ale z výše uvedených důvodů proběhla jen částečná implementace testů, která zabrala velké množství času. Nakonec byl tento framework nahrazen frameworkem Pixi.js.

2.4 EaselJS

Jedná se o jednu z knihoven ze souboru CreateJS. Podle [10] mezi další knihovny od CreateJS patří SoundJS, TweenJS nebo PrealoadJS a všechny zjednodušují práci s HTML5 elementy. Jak vyplývá z [11], EaselJS slouží k jednodušší a efektivní práci s elementem Canvas (plátno). Dokáže pracovat se zobrazenými elementy na plátně jako se zanořenými objekty. Jak je zmíněno v [1], na rozdíl od ostatních frameworků sám o sobě neobsahuje práci se zvuky nebo kolizemi. Proto nemusí být příliš vhodný, obzvláště pro vývoj složitějších her.

2.5 Phaser 3

Open-source framework pro vývoj převážně 2D her. Jak je zmíněno v [1], je založený na knihovně Flixel. Rozšiřuje práci s Canvas API a WebGL API. Na rozdíl od Unity nebo GDevelop neobsahuje žádná grafické prostředí a celý vývoj je založený na psaní kódu v jazyce JavaScript nebo TypeScript. Umožňuje uživateli efektivně pracovat se zvuky, animacemi, fyzikou herních objektů a dalšími.

Přestože Phaser pracuje s WebGL API, vývoj 3D projektů pomocí tohoto frameworku je zatím v ranné fázi (viz [12]). Bez extérního rozšíření (jako například Enable3D, jak je zmíněno v [13]) nenabízí možnost práce s 3D objekty, kamerami nebo světly. Tento nedostatek však vynahrazuje práce s 2D projekty, která je snadná, rychlá, rozsáhlá a pravidelně vylepšována aktivními vývojáři. Tento framework byl vybrán pro testování.

2.6 Unity

Herní engine, ve kterém lze vyvíjet multiplatformní 2D i 3D projekty, respektive hry. Jedná se o jeden z největších a nejpoužívanějších herních enginů na světě a díky tomu má velkou komunitu uživatelů a na internetu existuje spousta návodů, tutoriálů a rozsáhlá dokumentace. Podobně jako GDevelop, Unity má grafické prostředí, ve kterém lze upravovat vzhled herní úrovně, vkládat objekty, nastavovat jejich parametry a spoustu dalšího. Samotné chování hry probíhá pomocí uživatelských skriptů napsaných v jazyce C# nebo JavaScript. Ty se poté „připínají“ na objekty jako parametr a řídí jejich chování nebo logiku hry. Tento engine podporuje a nabízí herní možnosti jako simulace fyzikálního modelu světa, několik druhů kolizí, zvuky a další. 3D editoru samozřejmě nechybí kamery, světla nebo vykreslování stínů. Určitě stojí za zmínku, že Unity podporuje vývoj projektů ve virtuální realitě. [14]

Jak je ukázáno v [15], existuje několik verzí Unity. Základní osobní verze je zdarma, pokud na svých projektech uživatel vydělává méně jak sto tisíc dolarů ročně. Nabízí nejnovější verzi platformy a základní materiály pro začátek práce s Unity. Další osobní verze stojí patnáct dolarů měsíčně. Kromě základních vlastností nabízí také konzultace s Unity certifikovanými instruktory, další materiály pro práci s Unity a další. Následující dvě verze jsou určeny pro firmy. Při výdělcích pod dvě stě tisíc dolarů ročně se platí čtyřicet dolarů měsíčně, při větších výdělcích je nutné platit sto padesát dolarů měsíčně. Tyto verze nabízí tmavé pozadí uživatelského rozhraní, prémiové materiály pro práci platformou, prioritní přístup k Unity poradcům a další.

2.7 Pixi.js

Dle [1], Pixi.js se častěji využívá pro 2D vykreslování grafiky než pro přímý vývoj projektů, respektive her. To samozřejmě neznamená, že by pro vývoj herních projektů nebyl vhodný. Je velice oblíbený právě díky rychlému WebGL (nebo Canvas API) vykreslování grafiky. Pokud není WebGL podporováno, framework přejde na obyčejné Canvas API vykreslování. Pixi.js se často používá s jinými frameworky, nebo je jimi přímo používaný (Phaser, GDevelop…). Tento framework byl vybrán pro testování.

3 Výhody a nevýhody použití frameworků

Hlavním účelem frameworků je zjednodušit uživateli (respektive vývojáři) práci. Jak je zmíněno v [16], frameworky mají předefinované funkce, které lze použít a není potřeba znát jejich přesnou implementaci. Konkrétním příkladem složité implementace z herního průmyslu může být například fyzika herního prostředí. Existuje spousta frameworků pro vývoj her, které mají simulaci fyziky nebo jiné složité funkce implementovány, kdežto pokud by uživatel chtěl implementovat tyto funkcionality sám, zabralo by mu to desítky, možná stovky hodin, než by danou funkci implementoval a dovedl k dokonalosti. Použití frameworku tedy nejen zjednodušuje vývoj, ale zkracuje samotnou dobu vývoje, což je žádané.

Použití frameworku ale nemusí být vždy ten nejlepší nápad. Jak je psáno v [17], pokud vyvíjený projekt nevyžaduje složitější funkcionality, není potřeba používat framework. Samotný programovací jazyk JavaScript na rozdíl od použití frameworku nezabírá žádné místo a výsledná aplikace bude mít rychlejší načítání.

3.1 Použití frameworků vůči přímému použití WebGL/Canvas API

JavaScriptové frameworky pro vývoj her pracují s Canvas elementem a velmi často používají WebGL nebo Canvas API. Podle [18], Canvas API je navrženo právě tak, aby bylo využíváno frameworky. Výhody a nevýhody jsou podobné jako u používání samotného frameworku (viz kapitola 3). Pokud je vyvíjená aplikace (respektive hra) jednoduchá, není potřeba využívat frameworky a je možné použít pouze WebGL/Canvas API pro vykreslování. Pokud je potřeba implementovat složitější funkce (detekce kolizí, simulaci fyziky apod.), je vhodné využít framework za účelem zjednodušení práce a zkrácení samotné doby vývoje. [19]

Jedna z nejzákladnějších výhod použití frameworků oproti přímému použití HTML5 elementu Canvas je jednoduché vykreslení obrázků, k čemuž obvykle slouží třída Sprite implementovaná v jednotlivých frameworcích. Vloženému obrázku se mohou nastavit parametry jako velikost, pozice na plátně, rotace a podobně. Tyto parametry se dají nastavit dynamicky v běhu aplikace, tedy se jednoduše vytvářejí animace pohybu, rotace a další. K animaci změny textury se obvykle používá tzv. sprite sheet (viz. kapitola 5.4.2).

K dalším výhodám frameworků patří například detekce kolize nebo simulace fyzikálního modelu světa. Detekce kolize může být například mezi dvěma objekty (třídy Sprite), pokud mezi sebou kolidují, proběhne naprogramovaná akce, tedy kolizní efekt. Fyzikální model světa dokáže simulovat gravitaci, která působí na objekty, u kterých je tato funkce zapnutá. Také vypočítá směr odrazu (a provede odraz), pokud se dva objekty střetnou.

Frameworky obsahují spoustu dalších funkcí k usnadnění práce, v tomto případě usnadnění vývoje her. Mezi ně patří komfortnější práce se zvuky, vykreslováním nebo managementem scén (viz kapitola 5.4).

4 Způsob měření a návrhy testů

Pro implementaci byly vybrány frameworky Phaser, Pixi.js a BabylonJS. Při výběru byla nápomocná stránka z doporučené literatury (viz [1]), ale i samotné zadání a v něm zmíněné frameworky. Důvodem výběru právě zvolených frameworků byla absence grafického rozhraní. Ve spoustě případů může být takové rozhraní výhodou, ale také představuje výzvu v podobě učení se s novým prostředím.

Původně byl vybrán i framework ImpactJS, ale nakonec byl nahrazen frameworkem Pixi.js. Důvodem bylo hlavně nedostatek funkcí a možností (a tím způsobená časová náročnost implementace testů), tedy zastaralost ImpactJS. Po několika dnech návrhu řešení a implementace triviální funkce dynamické změny textu (která je ve většině novějších frameworků k dispozici) bylo dospěno k závěru, že tento framework bude nahrazen.

4.1 Měření výkonu

K měření výkonu jsem zvolil jednotku FPS z anglického frames per second, neboli snímky za sekundu. Jedná se o hodnotu, kolik snímků je program, respektive hra, schopná vykreslit za jednu sekundu. Jedná se tedy o jakousi jednotku plynulosti obrazu. Jak je zmíněno v [20], optimální hodnota v počítačové grafice je šedesát snímků za sekundu. Minimální počet, které je lidské oko schopné zachytit, aby obraz vypadal plynulý, je čtyřiadvacet FPS. Tato hodnota je však orientační, protože každý člověk má jiný zrak a plynulost obrazu se může zdát nepřesná už u hodnot pod třicet snímků za sekundu. Výstupem testů je tedy průměrná hodnota FPS.

Některé frameworky mají dostupné své vlastní počítadlo FPS (například BabylonJS), ale jiné zase ne. Z tohoto důvodu bylo nutné implementovat své vlastní řešení počítadla snímků za sekundu. Výhodou tohoto přístupu je, že pokud je použito vlastní řešení skrze všechny testované frameworky, udrží to konzistenci výsledků.

Při testování frameworku BabylonJS se ukazují hodnoty jak z vlastního počítadla FPS, tak z již dostupného z tohoto frameworku. Důvodem bylo zjištění, zda si BabylonJS nepřidává více snímků za sekundu ve vlastním počítadle FPS, aby vypadal lépe. Toto se však nepotvrdilo. Ve většině případů se výsledné hodnoty FPS z obou počítadel k sobě přibližují.

4.2 Navržené univerzální testy

Jedná se o testy, které je možné implementovat ve frameworcích podporující jak 2D grafiku, tak 3D z důvodu, že žádnou grafiku nevyužívají, nebo jsou oba typy grafik podporovány.

4.2.1 Test s kryptografickou hashovací funkcí

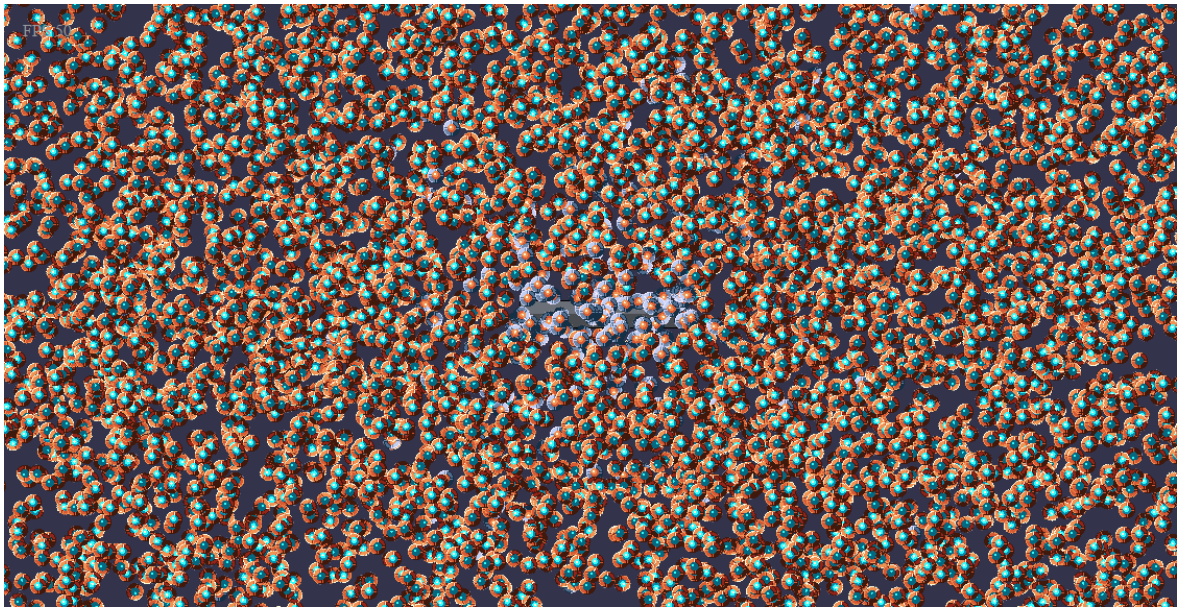
Obrázek 1 - Ukázka průběhu testu s kryptografickou hashovací funkcí (Pixi.js)

Hashovací funkce přijme jako vstup řetězec znaků a výstupem je zakódovaný řetězec pevné délky. Dle testu na [21] ze vstupu „Kobyla má malý bok“ můžeme dostat podobný výstup jako „180070568279434447afccc58afa213dcedb1074262a02976b84636bf602394e“ při použití typu hashovací funkce SHA256. Jak je zmíněno v [22], hashovací funkce jsou zamýšlené tak, aby text bylo možné zakódovat, nikoliv však dekódovat neboli aby nešlo zjistit zadaný vstup. Využívají se především při kódovaní uživatelských hesel. Zakódovaný výstup je v průběhu testu zakreslován jako text do plátna. Tento test ověřuje celkový výkon zařízení a také rychlost vykreslování textu na obrazovku. Test byl inspirován z Unity benchmarku (dostupné z [24]).

4.2.2 Test se zvukovými formáty

Tento test je jediný, jehož výstupem není průměrná hodnota snímků za sekundu, ale podporované zvukové formáty. Různé prohlížeče nepodporují všechny existující zvukové formáty a jednotlivé frameworky jich podporují ještě méně. Tento test přehraje zvuk pouze u podporovaných formátů. Jako testovaný zvuk bylo vybráno cinknutí mince (dostupné z [28]), které je používáno v mnoha hrách. [23]

4.2.3 Test s vytvářením 2D objektů



Obrázek 2 - Ukázka průběhu testu s vytvářením 2D objektů (Phaser)

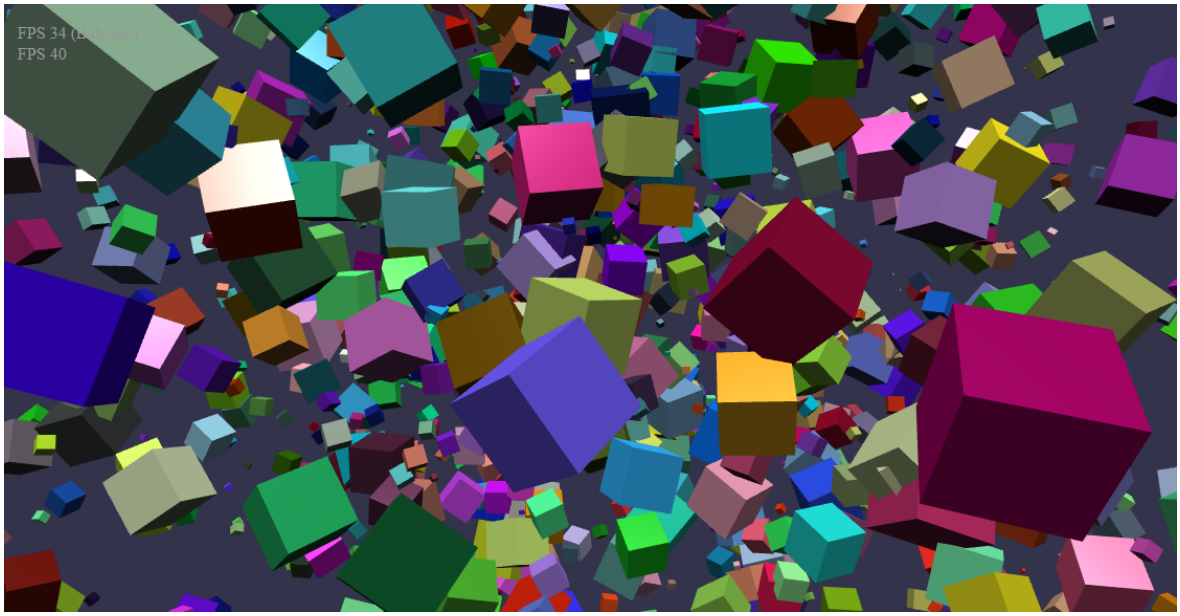
Tento test je poměrně rozsáhlejší než již dva zmíněné testy. V podstatě se jedná o vytváření 2D objektů (obrázků) s každým snímkem. Přes spuštěním tohoto testu je možné v aplikaci nastavit různé parametry, jako je počet vytvořených objektů za jeden snímek (hodnoty 1, 10, 30 až 100) a délka probíhajícího testu (od 10 sekund až po 2 minuty). Také samotné objekty mají nastavitelné vlastnosti, jedná se o animaci, rotaci, pohyb a detekci kolize. Výchozí pozice objektů je nastavena náhodně, avšak je omezená rozlišením obrazovky, aby se objekt neobjevil mimo zorný úhel, tedy by šel vidět.

Tímto způsobem lze zjistit, o jak moc výkonnostně náročnější je vytváření objektů bez vlastností než například objektů se zapnutou detekcí kolizí nebo animací. Tento test lze vytvořit i ve většině 3D frameworků, protože i do takových projektů lze vložit jednoduchý 2D obrázek. Díky tomu lze porovnat náročnost vytváření objektů mezi prostředím určeným k vývoji 2D projektů (Phaser a Pixi.js) a prostředím určeným vývoji k 3D projektů (BabylonJS).

4.3 Navržené testy v 3D prostředích

Jedná se o testy, které lze implementovat pouze ve frameworcích podporující 3D grafiku, tedy obsahují vykreslování 3D objektů, světel, stínů a podobně.

4.3.1 Test s vytvářením 3D objektů

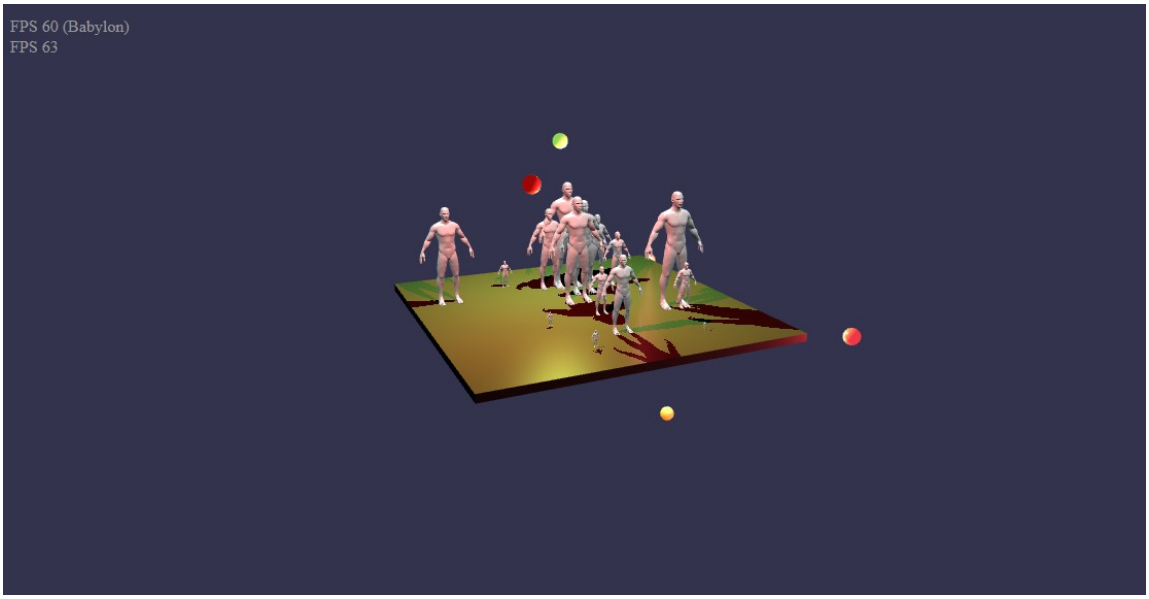


Obrázek 3 - Ukázka průběhu testu s vytvářením 3D objektů (BabylonJS)

Tento test je velice podobný testu s vytvářením 2D objektů. Opět se jedná o vytváření objektů s každým snímkem, kde počet vytvořených objektů za jeden snímek a samotná délka testu lze nastavit před samotným testem. Také lze nastavit různé vlastnosti vytvářených 3D objektů, a to jsou rotace, pohyb, detekce kolizí a simulace fyziky (tedy odrazů, gravitace a podobně). Ostatní vlastnosti jako pozice, velikost nebo barva jsou nastaveny náhodně (ale přiměřeně, aby se objekt neobjevit mimo zorný úhel kamery nebo aby jeho velikost nebyla příliš velká, popřípadě malá).

Jednotlivé 3D objekty se skládají z tzv. polygonů (mnohoúhelník). Dle [25] se jedná o jednoduchý geometrický útvar tvořený třemi nebo více stranami. Z těchto polygonů se následně skládají 3D modely. Pro představu, ze šesti pravidelných polygonů o čtyřech stranách (čtverců) lze složit 3D model kostky. Polygony určují náročnost vykreslení jednotlivých 3D modelů. Jednoduchá kostka lze vykreslit snadno, kdežto vykreslení kuličky je daleko výkonnostně náročnější. Proto u tohoto testu přibyla možnost nastavení typu vytvářeného objektu. Uživatel má na výběr ze tří, a to jsou kostka, kulička a model člověka (který je dostupný z [31]). Jejich ukázky jsou uvedeny v kapitole 5.5.

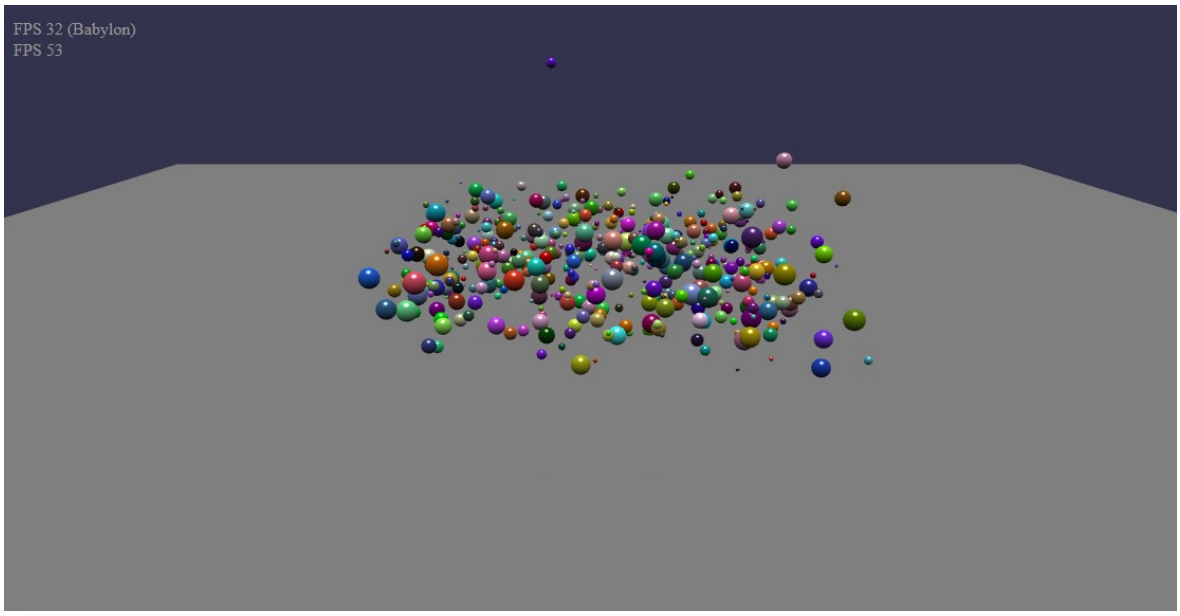
4.3.2 Test se světly a vykreslováním stínů



Obrázek 4 - Ukázka průběhu testu se světly a vykreslováním stínů (BabylonJS)

Přesně jak název vypovídá, jde o test, ve kterém se testuje výkon v závislosti na počtu světel a objektů vrhajících stín. Jedná se o jediný test, ve kterém je pohyblivá kamera. Ta krouží okolo téměř placatého kvádru reprezentující podložku, na níž jsou umístěny osvětlené objekty vrhající stín. Na tomto kvádru lze následně vidět vržené stíny objektů. Samotné světla se vytváří až v průběhu testu. Každé dvě sekundy se vytvoří jedno světlo, které následně začne kroužit okolo podložky. Před spuštěním testu lze nastavit délku testování, počet světel (jedno až deset), typ objektu vrhající stín (stejně jako u testu s vytvářením 3D objektů) a počet těchto objektů (5, 10, 15 až 30).

4.3.3 Test – padání kuliček



Obrázek 5 - Ukázka průběhu testu – padání kuliček (BabylonJS)

Tento test nelze nastavit před spuštěním testu(na rozdíl od přechozích zmíněných testů) a trvá třicet sekund. Před začátkem testu se vytvoří pět set kuliček o náhodné pozici a velikosti (ale

zároveň přiměřené, aby šly vidět nebo nebyly příliš velké). Také se vytvoří podložka, podobně jako u testu se světly, od které se kuličky při kolizi odrazí. Při spuštění testu se všem kuličkám aktivuje fyzikální model světa, takže začnou padat. Tento test dokáže zjistit, jak moc dokáže ovlivnit výkon výpočet směru odrazu pro každou kuličku v kolizi, ať už se jedná o kolizi se podložkou, nebo s jinou kuličkou.

5 Implementace testů a vytvoření instalačních balíčků apk

Kapitola pojednává o implementaci testů a jejich jednotlivých částí, taktéž i o implementaci měření výkonu, tedy snímků za sekundu. Konec kapitoly se zabývá exportem do apk formátu, tedy do instalačního balíčku pro operační systém Android, pomocí Apache Cordova.

5.1 Implementace měření snímků za sekundu

Byla vytvořena třída, která obsahuje tři metody. První slouží k výpočtu aktuální hodnoty snímků za sekundu. Její návratová hodnota je textový řetězec ve tvaru „FPS + ‚vypočtená zaokrouhlená hodnota FPS‘ “. Je volána ve všech testech (kromě testu se zvukovými formáty), respektive testovacích scénách pro zobrazení aktuální hodnoty snímků za sekundu. Ukázka této metody se nachází níže. Logika výpočtu je převzatá z [26]. Zatímco v první metodě proběhne výpočet, druhá metoda slouží pro ukládání současné hodnoty FPS do pole. Toto ukládání je oddělené od první metody, zobrazení FPS může probíhat i mimo dobu testování, kdežto ukládání hodnot probíhá pouze během testování. Poslední metoda má za úkol vypočítat průměrnou hodnotu FPS pomocí hodnot uložených v poli z druhé metody. Je volána po skončení testu.

```
getFPS() { //metoda pro výpočet, navrací řetězec i s hodnotou
        var delta = (Date.now() - this.lastCalledFPS) / 1000;
        this.lastCalledFPS = Date.now();
        this.currentFPS = 1/delta;
        this.FPStext = "FPS " + Math.round(this.currentFPS);
        return this.FPStext;
}
```

5.2 Implementace testu s kryptografickou hashovací funkcí

Byla napsána funkce, která vygeneruje náhodný text o předané délce parametrem (ukázka kódu níže), který se následně předá hashovací funkci (nebo metodě, záleží na typu implementace podle frameworku). Výstup neboli zakódovaný text je následně v jednotlivých frameworcích vypsaný na obrazovku společně s délkou vstupu. Samotný test trvá třicet sekund a obě tyto funkce jsou volány každý snímek (tedy v ideálním případě šedesát krát za sekundu). Délka náhodného vstupu začíná na hodnotě tisíc znaků a s každým snímkem se o další tisíc iteruje. Na konci testu se vypíše výsledek. Logika hashovací funkce je převzatá z [27].

```
var charactersForRandomText = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwx
yz0123456789';

var charactersForRandomTextLength = charactersForRandomText.length;
function randomText(length) { //funkce navrací náhodný textový řetězec
        var result = '';
        for ( var i = 0; i < length; i++ ) {
                result += charactersForRandomText.charAt(
                        Math.floor(Math.random() * charactersForRandomTextLength)
```

```
                );
            }
            return result;
}
```

5.3 Implementace testu se zvukovými formáty

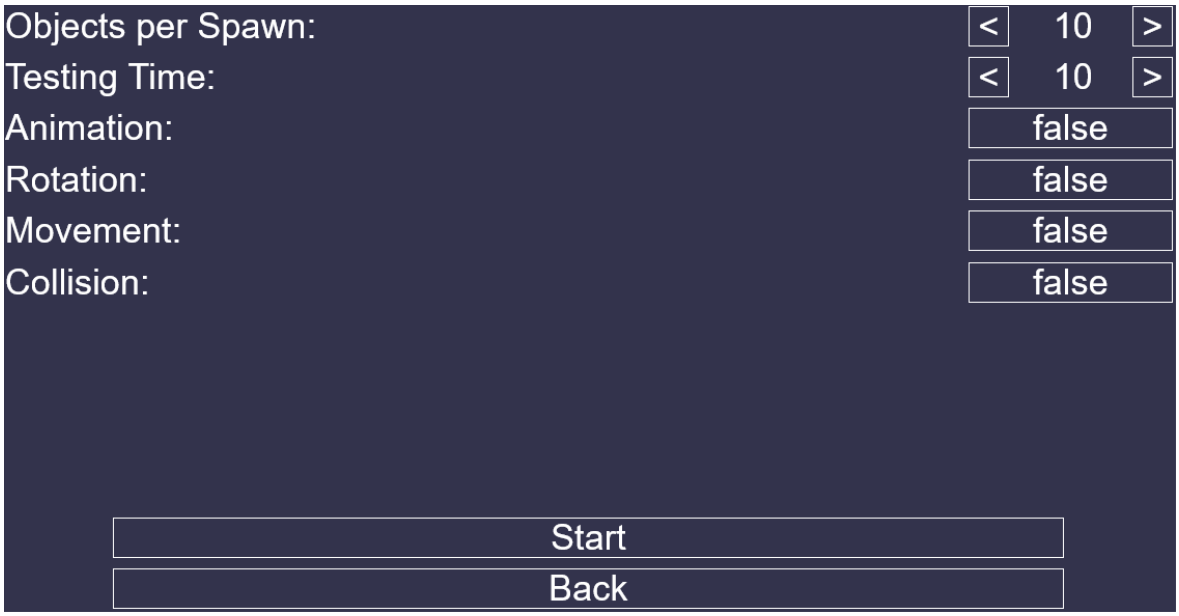
Vybraný zvuk (cinknutí mince) byl vyexportován do 31 různých formátů pomocí programu Switch Sound File Converter (dostupné z [29]), tedy je testováno celkem dvaatřicet různých formátů i s původním formátem (mp3). Nejdříve jsou formáty načteny do paměti. Následně jsou přehrávány po půl sekundových intervalech, celý test tedy trvá šestnáct sekund. Pokud formát není podporován, jeho zvuk není jednoduše přehrán.

Pixi.js jako jediný z vybraných frameworků nemá sám o sobě podporu přehrávání zvuků. Avšak bylo použito rozšíření PixiJS Sound (dostupné z [30]), který to umožňuje. Důvodem je, že tento framework je využívaný hlavně pro své vykreslování, proto není potřeba, aby bylo ve výchozí verzi zabudované přehrávání zvuků.

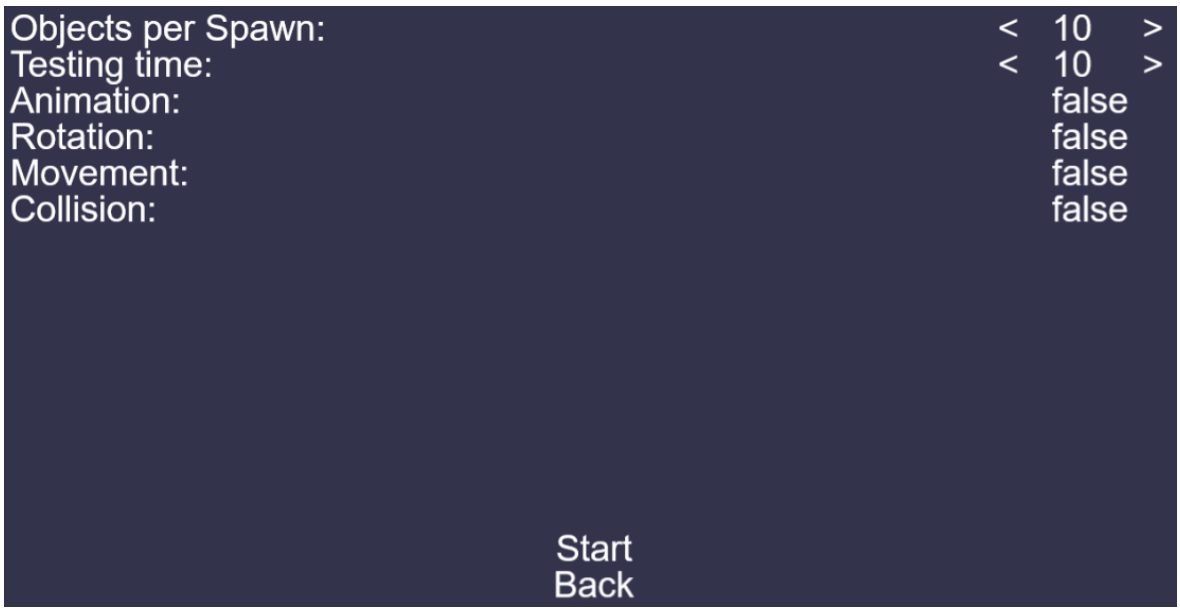
5.4 Implementace testu s vytvářením 2D objektů a konfigurační scény

Scéna je pojem používaný pro kontejner s více objekty (obrázky, uživatelské rozhraní a jiné). Dá se nad ní uvažovat jako nad herní úrovní a obvykle je tak i používána. Je možné mít definovaných více scén najednou, ale pouze jednu z nich vykreslovat. Díky scénám lze jednoduše přepínat mezi obsahem aplikace. Konkrétním příkladem můžou být právě zmíněné herní úrovně, nebo v tomto případě scéna pro menu a scéna pro testování.

5.4.1 Implementace konfigurační scény



Obrázek 6 - Scéna s možností konfigurace testu s vytvářením 2D objektů (BabylonJS)



Obrázek 7 - Scéna s možností konfigurace testu s vytvářením 2D objektů (Pixi.js)

Obrázek 8 - Scéna s možností konfigurace testu s vytvářením 2D objektů (Phaser)

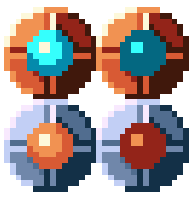
Jak již bylo zmíněno, test s vytvářením 2D objektů je nastavitelný, a proto bylo potřeba implementovat konfigurační scénu (viz Obrázek 6), ke které se bude přistupovat před samotným testem (který má svou vlastní scénu). Pro každou nastavitelnou hodnotu byla vytvořena globální proměnná s výchozí hodnotou deset nebo „false“ (podle typu proměnné). Na levé straně scény byly vytvořeny statické texty, které slouží jako popisky. Na pravé straně byly vytvořeny dynamické texty značící aktuální nastavenou hodnotu (dané globální proměnné). U hodnot, kde lze nastavit pouze „true“ nebo „false“, se po interakci (kliknutí nebo doteku) nastaví hodnota opačná tzn. z „true“ na „false“ nebo z „false“ na „true“. Tohle nastavení platí pro hodnoty značící animaci, rotaci, pohyb a kolizi. Pro každou nastavitelnou číselnou hodnotu, tedy pro počet vytvořených objektů za snímek a celkový čas testování, byly vytvořeny dvě šipky, které po interakci zvýší nebo sníží aktuální nastavený počet o hodnotu deset.

Vzhled uživatelského grafického prostředí byl implementován tak, aby vypadal stejně nebo velice podobně v každém použitém frameworku. Pokud porovnáme obrázky (Obrázek 6, Obrázek 7 a Obrázek 8), jde vidět, že tomu tak je. Mírné vzhledové rozdíly jsou jsou zapříčiněny způsobem vykreslování textu jednotlivými frameworky. Pokud by byl vzhled grafického prostředí složitější a nebo v každém frameworku jiný, mohlo by to mít vliv na výkon a způsobilo by to nežádoucí zkreslení výsledných hodnot. Proto jsou vzhledy použitých prostředí jednoduché a podobné.

5.4.2 Implementace testu s vytvářením 2D objektů

Byla implementována třída, jejíž jedna instance reprezentuje vytvořený 2D objekt. Vytvořenému objektu jsou zde nejen přiděleny parametry podle nastavených hodnot, ale také vygenerovány náhodné pozice podle rozlišení obrazovky. Níže je zobrazen konstruktor takové třídy, konkrétně ve frameworku Phaser.

```
constructor(scene, basicAnimState, collider) {

        this.object = scene.physics.add.sprite(16,16, "power-up");
        this.object.setRandomPosition(0, 0, config.width, config.height);

        //animation
        this.basicAnimState = basicAnimState;
        this.object.play(this.basicAnimState);

        //rotation
        if(rotation2D) this.rotation = Phaser.Math.Between(-1, 1)

        //movement
        if(movement2D) this.object.setVelocity(Phaser.Math.Between(-
        100, 100), Phaser.Math.Between(-100, 100));

        //collision
        if (collision2D) {
                scene.add.existing(this.object);
                scene.physics.world.enableBody(this.object);
                scene.physics.add.overlap(this.object, collider, this.collision.bind(this), null, scene);
        }
}
```



Obrázek 9 - Sprite sheet vytvářeného 2D objektu

Animace byla řešena pomocí tzv. sprite sheet, neboli jeden obrázek složený z více malých obrázků, které v rychlém sledu vytvářejí animaci. Jak ukazuje Obrázek 9, pokud se budou nekonečně velkou rychlostí střídat dva první obrázky, vytvoří to animaci blikání. Pokud má být animace zapnutá, je použito zmíněné „blikání“. Pokud je animace vypnutá, vytvoří se pouze první snímek z obrázku, tudíž statický obrázek bez animace.

Implementace rotace byla snadnější oproti animaci. V každém frameworku (ve kterých byl implementovaný tento test) lze nastavit hodnotu reprezentující úhel objektu. Pokud je tato hodnota nekonečně iterována, vytvoří to rotaci. Pokud má být rotace zapnutá, do proměnné reprezentující hodnotu přičítanou k úhlu, se vygeneruje náhodné číslo (mezi -1 až 1). Náhodnost hodnoty způsobí, že se každý objekt otáčí jinak rychle doleva nebo doprava. Pokud je rotace vypnutá, hodnota přičítána k úhlu je rovna nule.

Pohyb byl řešen dvěma způsoby. Některé frameworky (například Phaser) mají možnost nastavit rychlost objektů. V tomto případě stačí pouze nastavit náhodné hodnoty rychlosti na ose X a na ose Y. Náhodnost způsobí, že se každý objekt pohybuje jinou rychlostí a jiným směrem. Pokud pohyb nemá být zapnutý, není potřeba nic nastavovat, protože rychlost objektů má výchozí hodnotu nula, takže se nehýbe. Jiné frameworky nemají možnost nastavit rychlost (například Pixi.js), ale obvykle lze nastavit pozici objektu. Proto jsem použil stejné řešení jako u rotace, zkrátka vygenerování náhodných čísel, které se s každým snímkem přičítají k pozici na ose X a Y, čímž se vytvoří pohyb. Pokud je pohyb vypnutý, místo náhodné hodnoty se přičítá se nula, tedy se objekt nepohybuje.

Poslední parametr kolize byl nejkomplikovanější. Nejdříve byl nakreslen ležatý obdélník, který měl původně sloužit jako tlačítko pro grafické rozhraní. Později byl použit zde jako kolizní objekt. Ve chvíli, kdy vytvořený objekt překrývá kolizní objekt, je jeho barva trvale změněna na šedou, respektive se nastaví třetí obrázek ze sprite sheet jako výchozí (a to i když je zapnutá animace). Pokud nastavení kolize není zapnuté, kolizní objekt se jednoduše nevytvoří.

V Phaseru existuje přímo metoda overlap (použita v ukázce kódu výše), do které se jako parametry pošlou oba kolizní objekty a funkce, která se má vykonat při překrytí obou objektů. Tohle řešení bylo nejjednodušší a nejpříjemnější ze všech frameworků.

Ve frameworku pixi.js tato možnost nebyla, proto bylo potřeba vytvořit metodu, která kontroluje překrytí. Parametry jsou kontrolované objekty (vytvořený a kolizní). Díky pozici objektu, velikosti objektu a složitější podmínce (viz ukázka kódu níže) dokážeme zjistit, zda se překrývají. Pokud ano, provede se kolizní efekt (již zmíněné „přebarvení na šedou barvu“).

```
collisionCheck(a, b)

{
        if (this.object.textures != powerUpAnimStages.col) {
                var aa = a.getBounds();
                var bb = b.getBounds();
                if (aa.x + aa.width > bb.x &&
                   aa.x < bb.x + bb.width &&
```

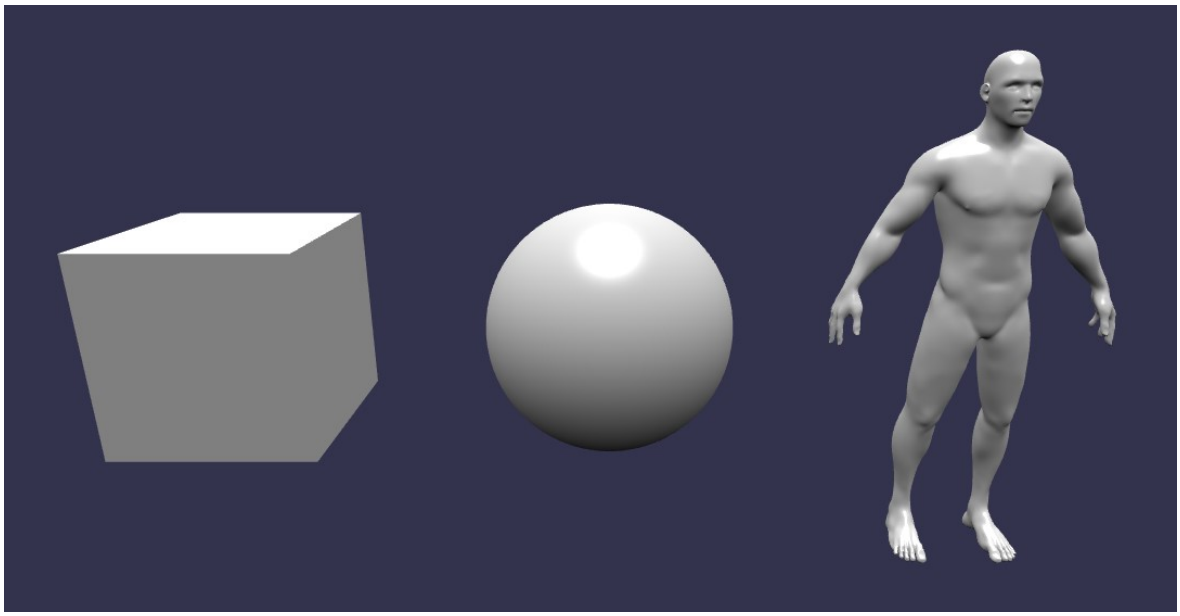
```
                    aa.y + aa.height > bb.y &&
                    aa.y < bb.y + bb.height) {
                        this.object.textures = powerUpAnimStages.col;
                        this.object.play();
                }
        }
}
```

Ve frameworku BabylonJS nebylo možné využít řešení, které bylo použito v Pixi.js. Důvodem je, že ve 3D scénách se nepoužívají pro určování pozic pixely jako ve 2D scénách. Ani řešení z Phaseru nebylo možné použít, protože 2D objekt, respektive třída Sprite, nemá v Babylonu metodu pro kontrolu překrytí. Avšak 3D objekt tuto metodu obsahuje, proto bylo implementováno řešení, které zahrnuje využití krychle. Pokud je kolize zapnutá, vytvoří se 3D kolizní objekt (velký modrý nehybný 3D kvádr uprostřed scény). Dále se s každým vytvořeným 2D objektem vytvoří také průhledná (neviditelná) krychle, která je „připnutá“ na 2D objekt (mají stejnou pozici) a má podobnou velikost jako vytvořený 2D objekt. U této krychle se kontroluje, jestli není v kolizi, tedy zda se nepřekrývá s kolizním objektem. Pokud ano, je opět proveden kolizní efekt („změna“ barvy na šedou). Může se stát, že je zapnutá detekce kolize a k tomu i pohyb, proto je potřeba s každým snímkem aktualizovat i pozici krychle na pozici vytvořeného 2D objektu. Pokud je kolize vypnutá, nevytváří se ani kolizní objekt, ani neviditelná krychle.

Při implementaci tohoto řešení nastal problém, kdy všechny vytvořené objekty byly vyhodnoceny jako v kolizi. Bylo to způsobeno tím, že neviditelná krychle se při svém vytvoření objeví na místě, kde je i kolizní objekt. Z toho vyplývá, že se při kontrole kolize spustí kolizní efekt. Pozice krychle byla samozřejmě nastavena na pozici 2D objektu okamžitě po jejím vytvoření, ale framework BabylonJS nestíhal aktualizovat pozici krychle před kontrolou kolize, proto byl spuštěn kolizní efekt, tedy všechny vytvořené 2D objekty byly šedé. Tento problém byl vyřešen použitím async a await, díky čemuž se nedříve počká na přesunutí krychle na správnou pozici, a až potom se vyhodnotí, zda je v kolizi. Použití async/await nijak neovlivnilo výkon.

Před začátkem samotného testu byl implementován ještě odpočet, který trvá tři sekundy. Je zde z důvodu, že pokud by se začaly vytvářet objekty hned po vstupu testovací scény, zkreslilo by to několik prvních uložených hodnot FPS, ze kterých se následně vypočítá průměrná hodnota FPS neboli výsledek testu. Po uplynutí odpočtu začne test. Po navolený čas se s každým snímkem vytvoří nastavený počet 2D objektů s vybranými parametry. Po uplynutí doby testování se zobrazí výsledek, vytvoří se tlačítko s funkcí návratu do menu a všechny vytvořené objekty jsou smazány. Důvodem mazání je, že na konci testu se obvykle hodnoty snímků za sekundu pohybují velice nízko, proto se celá scéna vykresluje trhaně. Pokud by tam objekty zůstaly, je dokonce velice pravděpodobný obtížný návrat do předchozí scény z důvodu dlouhého načítání. Po smazaní objektů je FPS opět na optimální hodnotě a návrat do menu je okamžitý po stisknutí vytvořeného tlačítka, což je více uživatelsky přívětivé.

5.5 Implementace testu s vytvářením 3D objektů



Obrázek 10 - Typy objektů, které je možné použít v testu s vytvářením 3D objektů (BabylonJS)

Jak bylo zmíněno v kapitole 4.3.1, tento test je funkcionálně podobný testu s vytvářením 2D objektů. Před samotným testem je zde opět konfigurační scéna, změnilo se ovšem několik parametrů. Přibyla simulace fyziky (odrazy a gravitace) a typ objektů (kostka, kulička a model člověka, viz Obrázek 10) a nastavení animace bylo odstraněno, ostatní parametry zůstaly stejné. Opět je vytvořená třída reprezentující jeden vytvořený 3D objekt. Jsou zde nastavovány parametry, ale také náhodné hodnoty pozic a nově i velikostí a barvy. Níže je zobrazen konstruktor takové třídy z frameworku BabylonJS.

```
constructor(index, scene) {

        if(objectType == "human") {
                this.object = human.clone();
                if(physics) {
                        this.object.physicsImpostor = new BABYLON.PhysicsImpostor(this.object, B
                        ABYLON.PhysicsImpostor.MeshImpostor, { mass: 2 }, scene);
                }
                this.object.scaling = randomVector3SameValue(0.01, 0.4);
                scene.meshes.push(this.object);
        }
        else if(objectType == "box") {
                this.object = BABYLON.MeshBuilder.CreateBox(
                "box"+index.toString(), {}, scene);
                if(physics) {
                        this.object.physicsImpostor = new BABYLON.PhysicsImpostor(this.object, B
                        ABYLON.PhysicsImpostor.BoxImpostor, { mass: 2 }, scene);
                }
                this.object.scaling = randomVector3SameValue(0.1, 2);
        }
```

```
        else if(objectType == "sphere") {
                this.object = BABYLON.MeshBuilder.CreateSphere(
                "sphere"+index.toString(), {}, scene);
                if(physics) {
                        this.object.physicsImpostor = new BABYLON.PhysicsImpostor(this.object, B
                        ABYLON.PhysicsImpostor.SphereImpostor, { mass: 2 }, scene);
                }
                this.object.scaling = randomVector3SameValue(0.1, 3);
        }
        this.object.position = randomVector3(-10, 10, -10, 10, -10, 10);
        this.object.material = new BABYLON.StandardMaterial(
        "material"+index.toString(), scene);
        this.object.material.diffuseColor = randomColor3();

        if(rotation) {
                this.rotation = randomVector3(-10, 10, -10, 10,
                -10, 10);
        }
        if(movement) {
                this.movement = randomVector3(-10, 10, -10, 10,
                -10, 10);
        }
}
```

Jak je uvedeno v [32], BabylonJS sám o sobě nemá možnost vložení vlastního 3D modelu, ale má knihovnu, která umožňuje vložit 3D modely několika formátů (konkrétně glTF, glb, obj a stl). S použitím této knihovny byl vložen do scény hlavního menu model člověka, který je následně přesunut mimo zorné pole kamery, ale stále existuje. Díky tomu je možné tento model naklonovat a vložit do jiné scény (tedy testovací scény) bez nutnosti ho znova načítat ze souboru při každém snímku. Samotný model má jiné měřítko než kostka a kulička, proto je nutné nastavovat jiné náhodné hodnoty velikostí, pokud je tento typ objektu při testování vybrán (příklad klonování a nastavování velikostí je v ukázce kódu výše).

Princip řešení rotace a pohybu zůstal stejný. S každým snímkem se objekt otočí/pohne náhodným směrem, tentokrát je ale rychlost otáčení nebo pohybu statická. Ve frameworku BabylonJS byly k tomuto použity metody „rotate“ a „translate“. Obě přijímají 3 parametry. První určuje směr rotace/pohybu (pokud je některý z parametrů vypnutý, předává se zde prázdný vektor, díky čemuž se objekt nehýbe/nerotuje). Druhým parametrem je rychlost rotace/pohybu a poslední parametr určuje, zda se pohyb/rotace koná z pohledu os objektu nebo os světových. Vždy jsou použité světové osy.

I princip detekce kolizí je velice podobný. Uprostřed scény se nachází kolizní objekt modré barvy. Pokud je detekce zapnutá, všechny vytvořené objekty mají zelenou barvu a při kolizi se změní červenou (kolizní efekt). Pokud není zapnutá, kolizní objekt není vytvořen a barvy objektů jsou nastaveny na náhodné. Detekce kolize se kontroluje s každým snímkem, zda se objekty nepřekrývají pomocí metody „intersectsMesh“ (ve frameworku BabylonJS).

Nastavení možnosti simulace fyziky bylo nejkomplikovanější. Nejdříve bylo potřeba aktivovat fyzikální model světa v celé scéně. Pokud není řečeno jinak, automaticky se tímto nastaví i gravitační zrychlení (konkrétně tedy hodnota – 9.823 na ose Y). Poté bylo potřeba nastavit kolizní zónu vytvářených objektů, což je jakýsi neviditelný tvar ohraničující viditelný objekt. Pokud se jedna kolizní zóna dotkne druhé, je simulován odraz. Objekty mohou mít různé tvary kolizních zón. Velmi často je v praxi využívána kolizní zóna tvaru obyčejné krychle z důvodu jednoduchosti, tedy malé náročnosti na výkon. U krychle je ovšem minimální přesnost, takže pokud je tato kolizní zóna nastavená u složitějšího modelu, třeba použitý 3D model člověka, může dojít k odrazu, i když se dva objekty viditelně nedotknou. Je také možné nastavit jiné tvary zón, například tvar kapsle. Nejsložitější a nejpřesnější je kompletní zmapování daného objektu a vytvoření kolizní zóny stejného tvaru jako má mapovaný objekt. Jednoduše řečeno, čím přesnější kolizní zóna, tím větší vytížení výkonu. V tomto případě, pokud je typ objektu nastaven na krychli, je kolizní zóna krychle. Pokud kulička, kolizní zóna je kulička. A pokud model člověka, je nastavená na celé zmapování.

Test probíhá stejně jako u testu s vytvářením 2D objektů. Nejdříve začne odpočet, poté proběhne test a po navolenou dobu se vytvářejí objekty s nastavenými parametry. Po uplynutí doby testu se všechny objekty zničí a zobrazí se výsledek.

5.6 Implementace testu se světly a vykreslováním stínů

Nejdříve byla potřeba implementovat konfigurační scénu. Princip implementace této scény se opět nijak neliší od předchozích konfiguračních scén. Možnosti nastavení jsou počet objektů vrhající snín, typ objektů vrhající stín, počet světel a testovací čas.

Na začátku byla vytvořena kamera, která krouží kolem testovací scény. Díky této funkci lze lépe vidět vržené stíny. BabylonJS má typ kamery právě k tomuto účelu, který jmenuje se „Arc Rotate Camera“. U této kamery lze nastavovat úhel natočení. Úhel je iterován přiměřenou hodnotou každý snímek a stejně tak je každý snímek nastaven kameře cíl pohledu na střed scény, ve kterém probíhá test. Výsledkem je rotující kamera zaměřená na střed scény.

Následně je vytvořen zvolený počet objektů (se zvoleným typem, tedy kostka, kulička nebo model člověka) vrhající stín. Těmto objektům je opět přidělena přiměřená náhodná pozice a velikost. Poté je vytvořen téměř placatý kvádr reprezentující zemi (dále zmiňované už jen jako země). U vytvořených objektů i země je nastaveno přijímaní stínu.

Na řadu přišla světla. Byla vytvořena třída reprezentující světlo. Světlu je přidělená náhodná barva a intenzita svítivosti. Každé světlo má také vlastní generátor stínů, do kterého je třeba vložit veškeré objekty, které budou vrhat stín působením daného světla. Dále byla implementována rotace světla kolem scény, respektive kolem země s objekty. S každým snímkem se pomocí jednoduché iterace a matematických funkcí sinus a cosinus nastaví pozice světla. Díky tomuto dokáže světlo opisovat přesnou kružnici okolo scény. Poloměr je nastavený přiměřeně náhodně, stejně tak hodnota iterace, která určuje rychlost rotace. Směr otáčení je vybrán náhodně z osmi směrů při vytvoření světla neboli v konstruktoru. Nakonec byla přidána kulička, která je „připnutá“ na vytvořené světlo. Její barva je nastavena na barvu daného světla a její pozice je s každým snímkem nastavena na pozici daného světla. Díky tomu lze lépe vidět aktuální pozici světla při průběhu testu. Ukázka konstruktoru této třídy je níže (BabylonJS).

```
constructor(scene, shadowCastingObjects, index) {

        var lightcolor = randomColor3();
        this.light = new BABYLON.PointLight("light"+index.toString(), new BABYLON.Vector3(0, 1
        , 0), scene);
        this.light.intensity = rndBetween(0.5, 1);
        this.light.diffuse = lightcolor;
        this.light.specular = lightcolor;

        this.lightSphere = BABYLON.Mesh.CreateSphere("sphere"+index.toString(), 10, 1, scene);
        this.lightSphere.position = this.light.position;
        this.lightSphere.material = new BABYLON.StandardMaterial("lightMat"+index.toString(), sce
        ne);
        this.lightSphere.material.emissiveColor = lightcolor;

        this.shadowGenerator = new BABYLON.ShadowGenerator(1024, this.light);
        for(var i = 0; i < shadowCastingObjects.length; i++) {
                this.shadowGenerator.getShadowMap().renderList.push(shadowCastingObjects[i]);
        }
        this.dirArr = rotateVariations[Math.round(rndBetween(0, rotateVariations.length-1))];
}
```

V průběhu testu je přidáno jedno světlo do scény každé necelé dvě sekundy, konktrétně 1,9 sekundy. Důvodem je, že pokud by test běžel 20 sekund (nejkratší možný nastavitelný čas) a mělo by se vytvořit 10 světel (největší nastavitelný počet světel), tedy jedno světlo každé dvě sekundy, došlo by k tomu, že se poslední světlo vytvoří právě ve dvacáté sekundě, kdy už by to nemělo smysl, protože ve dvacáté sekundě test končí. Když se každé jedno světlo vytvoří každé 1,9 sekundy, i poslední světlo vytvořené světlo bude mít vliv na výkon, protože se vytvoří v devatenácté sekundě, tedy ještě jednu sekundu bude ovlivňovat výkon.

Nejen v mobilním zařízení, ale i v prohlížeči na několikanásobně výkonnějším počítači se při vytvoření jednoho světla celá scéna zasekne. Proto se světla nevytvoří všechna světla naráz, ale vytvářejí se postupně v průběhu testu každé necelé dvě sekundy.

Před začátkem testu je samozřejmě implementován třísekundový odpočet. Po konci testu jsou světla a objekty vrhající stín smazány a je zobrazený výsledek spolu s tlačítkem „zpět“.

Ve frameworku BabylonJS nastal výrazný problém, co se týče světel. Jak je zmíněno v [33], výchozí nastavení materiálu (myšleno materiál, který určuje barvu nebo texturu 3D objektu) určuje, že maximální počet světel ve scéně je 4. Tuto hodnotu lze nastavit na vyšší. Vytvořil jsem tedy materiál, který je v mém testu společný pro objekty vrhající stín i kvádr reprezentující zemi, a tuto hodnotu jsem nastavil na 100. Avšak při vytváření mnoha světel bylo vidět, že už jedenácté vytvořené světlo nevytvářelo stíny. Řešením by mohlo být vykreslování nejbližších viditelných světel (respektive stínů). Toto řešení však není ideální vzhledem k tomu, že je žádoucí otestovat vetší počet světel ve scéně. Z tohoto důvodu je maximální počet světel v konfigurační scéně tohoto testu omezen na deset.

Zmíněný problém se světly může být výrazný obzvláště, pokud uživatel implementuje herní úroveň nebo celou hru odehrávající se v noci. Typickým příkladem může být závodní noční level na trati, která je osvětlována pouze pouličními lampami. V tomto případě je dokonce vysoce pravděpodobné, že je v jeden okamžik vidět více než deset světel.

5.7 Implementace testu – padající kuličky

Před začátkem testu je doprostřed scény vložen kvádr reprezentující zemi (dále zmiňované už jen jako země), podobně jako u testu se světly. Tentokrát je ovšem tato země mnohonásobně širší a delší. Také je vytvořeno 500 kuliček náhodné velikosti a barvy. Pozice kuliček je sice náhodná, ale zároveň omezená tak, aby se kuličky objevily nad zemí. Ve scéně je následně zapnutý fyzikální model světa, ale gravitační zrychlení je prozatím nastaveno na 0, takže se objekty (kuličky) nepohybují.

Opět následuje tří sekundový odpočet, po kterém se všem kuličkám nastaví kolizní zóna (ve tvaru kuličky). Ve stejném momentu se také zapíná gravitační zrychlení a kuličky začínají padat. Zapnutý fyzikální model světa způsobuje, že se kuličky odráží jedna od druhé a také od země. Po uplynutí třiceti sekund se kuličky zničí, zobrazí se výsledek testu spolu s tlačítkem „zpět“.

```
function startDropingTest(timeout, objects, physicsEngine, scene) { //funkce, která "zapne" u všech kuli
ček fyziku a nastaví gravitační sílu
   setTimeout(() => {
      for (var i = 0; i < objects.length; i++) {
         objects[i].enablePhysics(scene);
      }
      physicsEngine.setGravity(new BABYLON.Vector3(0, -9.81, 0));
      dropingSpheresRunning = true;
   }, timeout);
}
```

5.8 Export a implementace ukládání souboru

Celá práce je testovaná ve webovém prostředí, avšak samotný benchmark bude prováděn na mobilním zařízení. Toto zařízení má operační systém Android, proto je potřeba vyexportovat práci do instalačního souboru formátu apk. K tomuto byl použit Apache Cordova (viz [34]). Po nutné instalaci Node.js byly vytvořeny Apache Cordova projekty pro každý framework, ve kterém proběhla implementace. Do každého se také přidala platforma Android. S vytvořením projektu se automaticky vytvoří i soubor config.xml, který, jak jeho název vypovídá, slouží ke konfiguraci. Bylo zde pozměněno několik parametrů, které jsou informační, jako název nebo popis aplikace. Dále zde bylo přidáno omezení, které „zamkne“ orientaci obrazovky (převzato z [35]). Pokud by se totiž aplikace otočila, celé grafické rozhraní by se rozházelo, což je nežádoucí. Dle [36] jsou zde také připsány dva řádky kvůli zapsaní výsledků do souboru a uložení souboru. Tyto řádky zajistí vyžádání o povolení k přístupu k souborům, a pokud je následně povoleno, soubor se uloží. Nakonec byl pomocí jednoduchého příkazu vygenerován apk soubor.

Samotné ukládání souboru je implementováno právě přes Apache Cordovu. Nejdříve bylo potřeba nainstalat Cordova Plugin File, který slouží k ukládání souborů. Po vygenerování apk se automaticky vytvoří skript cordova.js, pomocí kterého je možné ukládat. Pokud je s ním žádaná práce, je potřeba ho zahrnout do používaných souborů ve výchozím html souboru. V tomto případě index.html. Následně se v kódu vybere místo uložení (adresář), pomocí třídy Blob se vytvoří textový soubor (formát txt) a jsou do něj zapsané výsledky, které jsou ukládané v průběhu testování do paměti. Tento soubor je následně uložen na zvolené místo. Implementace je převzatá z [37] a ukázka je níže.

```
async download() {

        var name = this.fileName;
        var text = this.resultsText;
        var filePath = "";

        await window.requestFileSystem(LocalFileSystem.PERSISTENT, 0, function (fs) {
                var fileDir = cordova.file.externalDataDirectory.replace(cordova.file.externalRootDire
                ctory, "");
                filePath = fileDir + name;
                fs.root.getFile(filePath, { create: true, exclusive: false }, function (fileEntry) {
                        fileEntry.createWriter(function (fileWriter) {
                                var dataObj = new Blob([text], {type: "text/plain"});
                                fileWriter.write(dataObj);
                        });
                });
        });
}
```

6 Testování a výsledky

Testy byly prováděny na dvou zařízeních. Jedná se o mobilní telefon Huawei P10 lite a tablet Lenovo, kde bylo použité webové prostředí, konkrétně prohlížeč Mozilla Firefox. Podrobnější specifikace zařízení uvádí Tabulka 1:

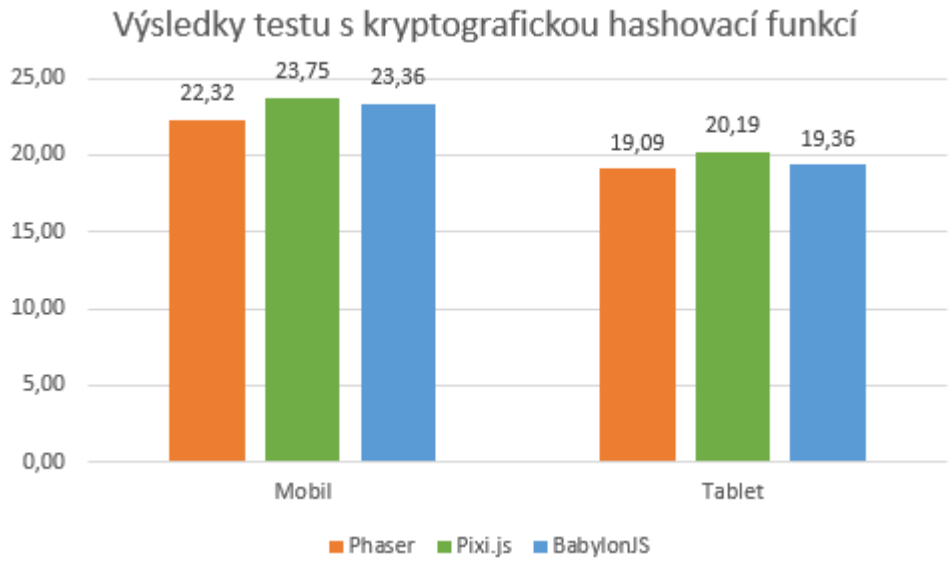
Tabulka 1 - Specifikace použitých zařízení

	Mobil	**Tablet**
Název	HUAWEI P10 lite	Lenovo
Operační systém	Android 8.0.0	Windows 8.1 (Desktop)
Procesor	HiSilicon Kirin 658 Octa-core (4 x 2,1 GHz & 4 x 1,7 GHz)	Intel(R) Atom(TM) CPU Z3735 @ 1,33 GHz
Operační paměť (RAM)	3 GB	2 GB
Počet jader	8	4
Grafický procesor (GPU)	Mali-T830 MP2	Inter(R) HD Graphics
Rozlišení obrazovky	1920 x 1080 pixelů	1024 x 768 pixelů

Každý test má implementovanou jednu nebo více náhodných funkcí (ať už se jedná o náhodou pozici či velikost, nebo generování náhodného textu). Kvůli tomu se při opětovných spuštěních výsledky testů nepatrně liší. Proto byl každý test spuštěn pětkrát a z výsledků je spočítána průměrná hodnota, tedy úplným výsledkem je průměrná hodnota z pěti výsledných hodnot snímků za sekundu.

Při testování bylo zjištěno, že první spuštění testu má výslednou hodnotu podstatně nižší než při dalších spuštěních. Důvodem je pravděpodobně nedostatečné prvotní načtení a předzpracování JavaScriptem, které se již při dalších spuštěních testů neskuteční. Nicméně, každý test (a každé jeho nastavení) byl nejdříve jednou nebo dvakrát spuštěn bez zapsání výsledků. Přesto je větší počet prvních zapsaných hodnot menší než u následujících, jak je vidět v přiložených souborech xlsx formátu, tedy ve výsledných tabulkách.

6.1 Výsledek testu s kryptografickou hashovací funkcí



Obrázek 11 - Graf výsledků testu s k. hashovací funkcí

Očekávaným výsledkem byly stejné nebo velice podobné hodnoty. Jak prezentuje Obrázek 11, předpoklad se stal skutečností. Výpočty hashovací funkce testují hlavně hardware zařízení. Mírný odklon hodnot mezi frameworky může být způsoben rozdílnými způsoby vykreslování textu, které frameworky využívají. Nejlepší hodnoty má v tomto testu má Pixi.js, a to jak na mobilním telefonu, tak na tabletu.

6.2 Výsledek testu se zvukovými formáty

Tabulka 2 - Přehled podporovaných/nepodporovaných zvukových formátů

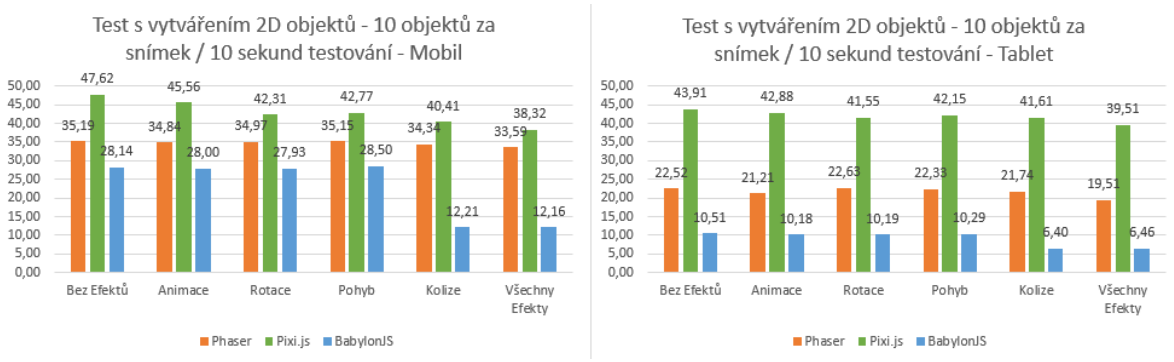
	Mobil			Tablet		
Zvukový formát	**Phaser**	**Pixi.js**	**BabylonJS**	**Phaser**	**Pixi.js**	**BabylonJS**
mp3	Ano	Ano	Ano	Ano	Ano	Ano
ogg	Ano	Ano	Ano	Ano	Ano	Ano
wav	Ano	Ano	Ano	Ano	Ano	Ano
acc	Ne	Ano	Ne	Ne	Ano	Ne
flac	Ne	Ano	Ne	Ne	Ano	Ne
mov	Ne	Ano	Ne	Ne	Ano	Ne
adts	Ne	Ano	Ne	Ne	Ne	Ne
m4a	Ano	Ano	Ne	Ano	Ano	Ne
m4b	Ne	Ano	Ne	Ne	Ano	Ne
m4r	Ne	Ano	Ne	Ne	Ano	Ne
opus	Ano	Ano	Ne	Ano	Ano	Ne

Jak ukazuje Tabulka 2, formáty mp3, ogg a wav jsou podporované ve všech testovaných frameworcích jak na mobilním telefonu, tak na tabletu, respektive v prohlížeči. Testované formáty, které nebyly podporované ani v jednom případě, Tabulka 2 neuvádí (z celkového počtu dvaatřiceti formátu bylo tedy jen 11 v některém případě přehráno). Jednotlivé úseky tabulky (mobil/tablet) se neliší až na formát adts, který je přehrán ve frameworku Pixi.js na mobilním telefonu, ale nikoliv na

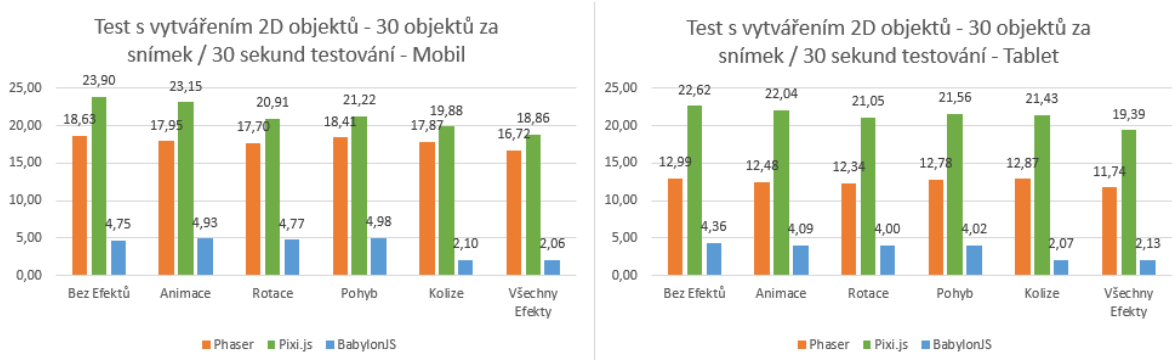
tabletu. Pravděpodobným důvodem je neposkytnutí knihovny pro dekódování operačním systémem. Ostatní výsledky odpovídají.

6.3 Výsledek testu s vytvářením 2D objektů

Pro tento test byly provedeny dvě různá nastavení z důvodu možnosti různé konfigurace. První nastavení bylo pro vytváření deseti objektů za jeden snímek po dobu deseti sekund (viz Obrázek 12). Druhé nastavení bylo pro třicet objektů za jeden snímek po dobu půl minuty (viz Obrázek 13). Obě nastavení byla provedena pro 6 různých kategorií, a to jsou animace, rotace, pohyb, kolize, bez efektu a se všemi efekty.



Obrázek 12 - Grafy výsledků testu s vytvářením 2D objektů - 1. konfigurace



Obrázek 13 - Grafy výsledků testu s vytvářením 2D objektů - 2. konfigurace

Obrázek 12 a Obrázek 13 ukazuje zejména jasné „vítězství“ frameworku Pixi.js. Jeho hodnoty se pohybují ve vysokých hodnotách oproti ostatním frameworkům. Na mobilním telefonu lze vidět, že každý efekt odebírá menší množství výkonu. Skok mezi kategoriemi bez efektů a všechny efekty je poměrně velký, ale i tak je jeho výkon všude největší. Na tabletu jsou jednotlivé rozdíly hodnot mezi kategoriemi o něco zanedbatelnější, však u pokles u všech efektů je také znát. Překvapivě se hodnoty mezi mobilním telefonem a tabletem zase tolik neliší.

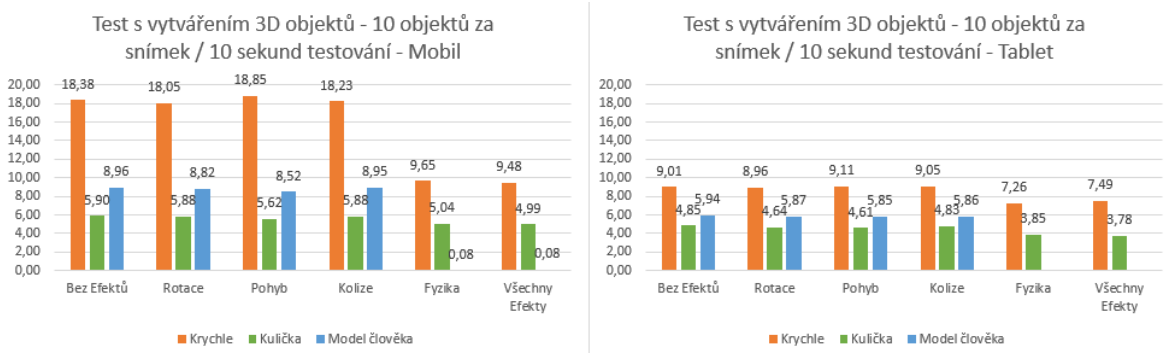
Phaser má o něco nižší hodnoty, avšak mnohem více konzistentní. Kromě menšího poklesu (zhruba 1 snímek za sekundu) u kategorie všechny efekty jsou rozdíly mezi jednotlivými kategoriemi zanedbatelné, což se dá považovat za velice pozitivní vlastnost. Na rozdíl od Pixi.js je zde však vidět veliký pokles mezi zařízeními.

Jak se dalo očekávat, framework BabylonJS, který je určený pro 3D projekty, byl v tomto testu nejslabší. Veliký pokles můžeme vidět zejména u kategorií kolize a všechny efekty (která zmíněnou kolizi využívá). Příčinou je způsob detekce kolizí zmíněný v kapitole o implementaci

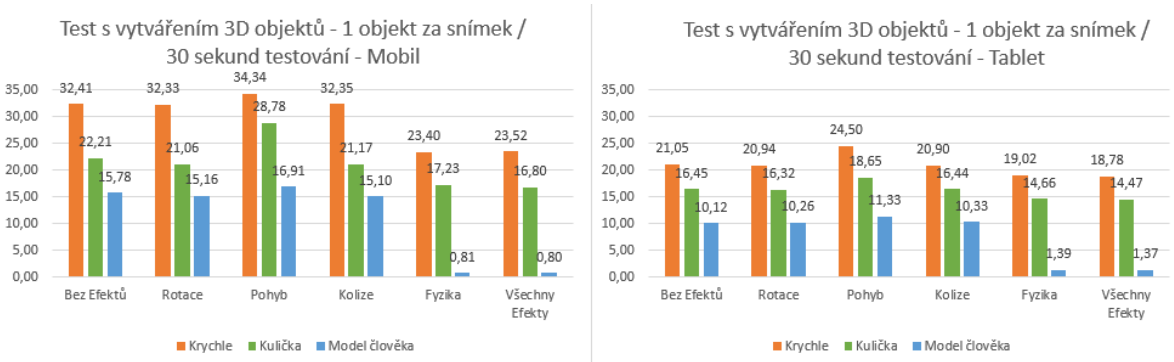
tohoto testu (kapitola 5.4.2). Rozdíly mezi ostatními kategoriemi jsou zanedbatelné. Odlišnosti mezi zařízeními jsou vidět pouze u první konfigurace. Tohle je způsobeno kratší dobou testování než u druhé konfigurace. Pokud by 1. nastavení trvalo delší dobu, pravděpodobně by hodnoty byly podobné.

6.4 Výsledek testu s vytvářením 3D objektů

Stejně jako u předchozího zmíněného testování jsou zde dvě různé konfigurace. První je opět vytváření deseti objektů za snímek po dobu deseti sekund. Následně bylo vyzkoušeno testování vytváření třiceti objektů po dobu půl minuty. Avšak tento test byl opravdu pomalý a výsledné hodnoty často nulové. V některých případech dokonce aplikace přestala pracovat (popřípadě se přestal načítat prohlížeč) z důvodu značného vytížení. Kvůli tomu se počet vytvořených objektů v druhé konfiguraci snížil ze třiceti na jeden. Druhá konfigurace je tudíž o něco méně náročná. Opět je tu 6 kategorií: rotace, pohyb, kolize, fyzika, bez efektů a se všemi efekty. Tento test byl prováděn pouze ve frameworku BabylonJS, proto tentokrát jednotlivé barvy sloupců (Obrázek 14 a Obrázek 15) reprezentují jednotlivé testované typy objektů, které je možné před spuštěním testu navolit.



Obrázek 14 - Grafy výsledků testu s vytvářením 3D objektů - 1. konfigurace



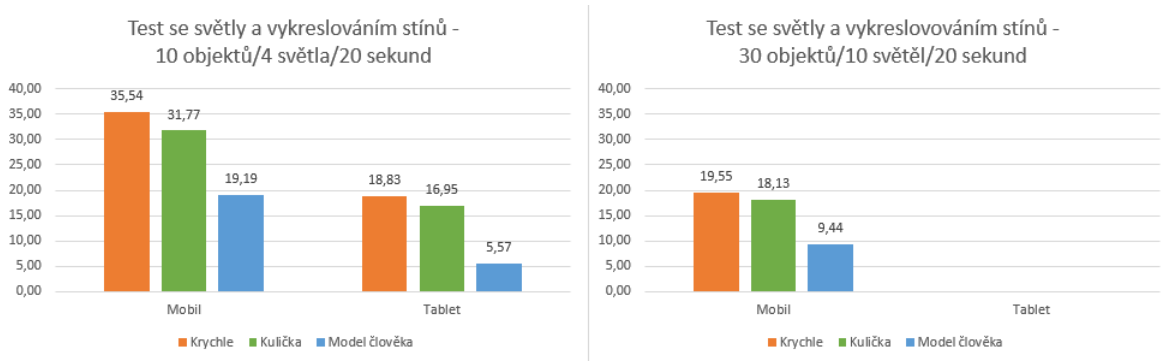
Obrázek 15 - Grafy výsledků testu s vytvářením 3D objektů - 2. konfigurace

Lze si povšimnout, že nejvyšší hodnoty jsou při použití typu objektu krychle. Důvodem je jeho nízký počet polygonů, konkrétně šest. Zvláštním úkazem jsou rozdílné hodnoty objektu kuličky a modelu člověka mezi oběma konfiguracemi. Pokud jsou ignorovány kategorie fyzika a všechny efekty, v první konfiguraci jsou hodnoty při použití kuličky menší než při použití modelu člověka, zatímco v druhé konfiguraci jsou větší než při použití modelu člověka, a to jak na mobilním telefonu, tak na tabletu. Framework pravděpodobně provádí nahrazení plochy kuličky sítí trojúhelníkových polygonů, jejichž počet může být ve výsledku větší než pro model člověka.

Kategorie pohyb má pro každý jednotlivý objekt nejvyšší hodnoty. Důvodem je, že objekt zanedlouho po svém vytvoření s velkou pravděpodobností vyletí (kvůli pohybu) ze zorného úhlu kamery, tedy pryč ze scény. Pokud je objekt pryč ze scény, není vykreslován, což značně ulehčí náročnosti na výkon. Tento jev byl očekáván i při testu s vykreslováním 2D objektů s kategorií pohyb, ale neprokázal se. Velký pokles byl zaznamenán v kategorii fyzika, nejvíce však při použití modelu člověka. Příčinou je vytvoření kolizní zóny, která je v tomto případě nastavená na kompletní zmapovaní celého objektu, proto je tak náročná (viz kapitola 5.5). V 1. konfiguraci při použití tabletu nebyly testy se simulací fyziky a modelem člověka z důvodu náročnosti dokončeny, proto nejsou hodnoty nastavené vůbec. Ostatní nezmíněné kategorie (rotace a kolize) nemají na výkon téměř žádný vliv (porovnáváme-li je s kategorií bez efektů).

6.5 Výsledek testu se světly a vykreslováním stínů

I u tohoto testu jsou dvě nastavení. První konfigurace (Obrázek 16, levá polovina) zahrnuje vytvoření deset objektů vrhajících stín, čtyři postupně se objevující světla a dobu trvání dvaceti sekund. Druhá konfigurace (Obrázek 16, pravá polovina) je nastavena na nejnáročnější možné hodnoty, tedy třicet objektů vrhajících stín, deset světel a opět po dobu dvaceti sekund.

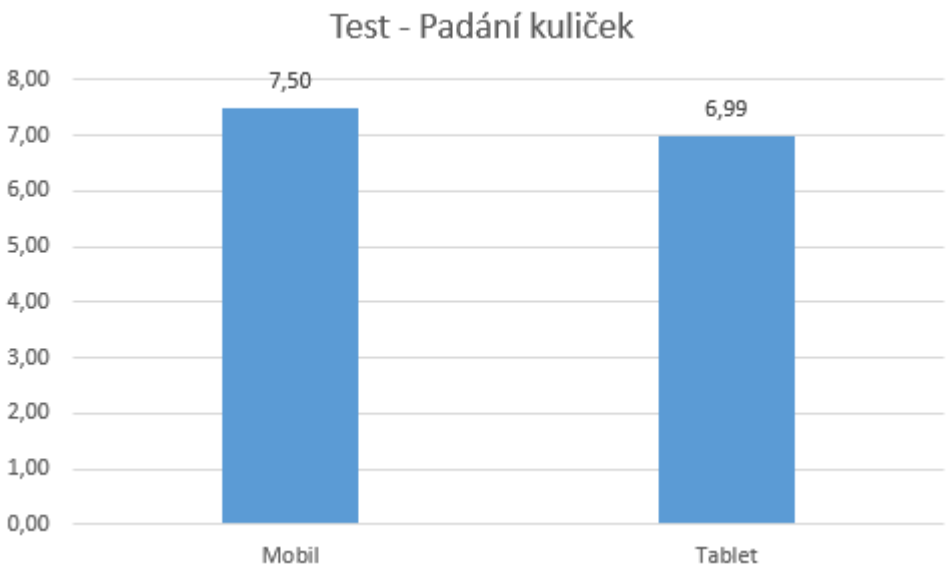


Obrázek 16 - Graf výsledků testu se světly a vykreslováním stínů

Nejlepší hodnoty má varianta s použitím krychle z důvodu jednoduchého tvaru, a tedy i následně vykresleného stínu. Test s objektem kulička má jen o trochu menší hodnoty, ale přesto má tento objekt složitější tvar než krychle, proto je zde znatelný pokles oproti prvnímu zmíněnému objektu. Nejhůře je na tom varianta s modelem člověka, kdy jeho stín je nejsložitější vykreslit, proto je vidět velký skok mezi hodnotami oproti testu s krychlí nebo kuličkou.

Tento test měl největší náročnost, čímž je myšleno, že platforma kvůli nedostatku výkonu nebo zátěži aplikaci vynuceně ukončovala. Důkazem je nemožnost naměřit hodnoty FPS v případě použití tabletu a druhé konfigurace. I přes mnoho pokusů tento test nikdy neskončil ukázáním výsledku, nýbrž trvalým zamrznutím obrazovky nebo pádem prohlížeče.

6.6 Výsledek testu – padající kuličky



Obrázek 17 - Graf výsledku testu – padající kuličky

Tento test probíhal pouze ve frameworku BabylonJS, takže Obrázek 17 prezentuje zejména výkonnostní rozdíl mezi mobilním telefonem a tabletem. Pokud je tento výsledek porovnán s výsledky testu s vytvářením 3D objektů (viz kapitola 6.4, Obrázek 14 a Obrázek 15) při použití objektu kulička a simulace fyziky, lze vidět, že samotné vytváření objektů má velký vliv na výkon. Pokud je tento test srovnán s první konfigurací testu s 3D objekty, jsou zde hodnoty větší, avšak při porovnání s druhou konfiguraci jsou zde hodnoty menší. To dokazuje, že pokud jsou objekty již vytvořeny ve scéně před měřením výkonu, je to méně náročnější než vytváření většího počtu objektů při měření výkonu.

7 Závěr

Testy ukázaly, že nejméně výkonnostně náročným použitým frameworkem je Pixi.js. Krom toho se s ním příjemně pracuje, avšak při tvorbě složitějších projektů, respektive her, nemusí být vhodný. Postrádá některé funkce, které například framework Phaser nabízí. Phaser sice není tak rychlý jako Pixi.js, avšak nabízí lepší management scén a mnohem více možností. Oba zmíněné frameworky jsou vhodné pro vývoj 2D her. Pokud porovnáme práci s těmito frameworky s prací ve frameworku ImpactJS, ve kterém proběhly pokusy o implementaci testů, jsou mnohem lepší, příjemnější, nabízí více možností a mají rozsáhlejší uživatelskou komunitu i dokumentaci. BabylonJS je skvělý framework pro 3D projekty, avšak už méně pro 2D. Testy ukázaly, že oproti ostatním testovaným frameworkům BabylonJS v 2D testech zaostává. Tento framework se určitě hodí pro různé vykreslování složitějšího 3D objektu a jeho prezentaci, ale už méně pro složitější 3D scény se spoustou světel a objektů. Pro vývoj nenáročných multiplatformních, respektive mobilních her je BabylonJS vhodný, ale na složitější 3D hry (nebo 2D hry s 3D prvky) z výkonnostních důvodů vhodný není. Dobrou náhradou za tento framework může být herní engine Unity, ve kterém sice neproběhly testy, ale z osobní zkušenosti ho mohu doporučit.

Na práci by se dalo navázat implementací dalších testů a jejich použití v dalších frameworcích. Současné testy byly vytvořeny pouze s pomocí frameworků a psaní kódu, tedy neproběhla žádná implementace pomocí herního enginu a grafického prostředí. Proto by bylo určitě dobré také vyzkoušet výkon herních enginů (jako je výše zmíněné Unity nebo GDevelop), zda vývoj pomocí grafického prostředí má na výkon vliv. Popřípadě porovnat, jak velký výkonnostní rozdíl bude mezi spuštěním aplikace přímo ve vývojovém prostředí (běžně používanému k hledání chyb a testování) a výslednou aplikací.

Práce pro mě byla velice přínosná, zejména jsem se naučil lépe pracovat s frameworky pro vývoj her, webovým prostředím ale i s programovacím jazykem JavaScript. Do budoucna se mi budou hodit zkušenosti s exportem do mobilní aplikace, a celkové práce s Apache Cordova. Samotné informace, které práce poskytla, myšleno výsledky testů a závěry ohledně frameworků, budou i pro mou osobu užitečné, jelikož bych se rád v budoucnosti věnoval vývoji her, minimálně jako hobby.

Literatura

1. HTML5 Game Engines. *HTML5 Game Engines* [online]. [cit. 2020-04-12]. Dostupné z: https://html5gameengine.com/
2. The game engine for everyone. *GDevelop* [online]. [cit. 2020-04-12]. Dostupné z: https://gdevelop-app.com/
3. GDevelop 5 Review. *Choose the best - Slant* [online]. [cit. 2020-04-12]. Dostupné z: https://www.slant.co/options/6630/~gdevelop-review
4. WebGL: 2D and 3D graphics for the web. *MDN web docs* [online]. 2020, 31 Mar 2020 [cit. 2020-04-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
5. Babylon.js Playground. *Babylon.js Playground* [online]. [cit. 2020-04-12]. Dostupné z: https://www.babylonjs-playground.com/
6. Use a Physics Engine. *Babylon.js Documentation* [online]. [cit. 2020-04-12]. Dostupné z: https://doc.babylonjs.com/how_to/using_the_physics_engine
7. ImpactJS Review. *Choose the best - Slant* [online]. [cit. 2020-04-12]. Dostupné z: https://www.slant.co/options/1061/~impactjs-review
8. Entity. *Impact – HTML5 Canvas & JavaScript Game Engine* [online]. [cit. 2020-04-12]. Dostupné z: https://impactjs.com/documentation/class-reference/entity
9. Font. *Impact – HTML5 Canvas & JavaScript Game Engine* [online]. [cit. 2020-04-12]. Dostupné z: https://impactjs.com/documentation/class-reference/font
10. CreateJS. *CreateJS* [online]. 2018 [cit. 2020-04-12]. Dostupné z: https://createjs.com/
11. EaselJS. *CreateJS* [online]. 2018 [cit. 2020-04-12]. Dostupné z: https://createjs.com/easeljs
12. Does phaserjs support 3d at all. *Stack Overflow* [online]. 2014, 15. 3. 2014 [cit. 2020-04-13]. Dostupné z: https://stackoverflow.com/questions/22418351/does-phaserjs-support-3d-at-all
13. Easily add 3D and 3D Physics to your Phaser games with this powerful modular plugin. *Phaser* [online]. 2020, 3 Apr 2020 [cit. 2020-04-13]. Dostupné z: https://phaser.io/news/2020/04/enable3d
14. Unity. *Unity* [online]. [cit. 2020-04-13]. Dostupné z: https://unity.com/
15. Plans and pricing. *Unity store* [online]. [cit. 2020-04-13]. Dostupné z: https://store.unity.com/
16. JavaScript Frameworks: To Use or Not To Use? *Noupe - THE magazine for webworkers and site owners* [online]. 2016, 26. Jan 2016 [cit. 2020-05-01]. Dostupné z: https://www.noupe.com/development/javascript-frameworks-94897.html
17. What are advantages of JavaScript frameworks over pure/native JavaScript? *Quora - A place to share knowledge and better understand the world* [online]. [cit. 2020-05-01]. Dostupné z: https://www.quora.com/What-are-advantages-of-JavaScript-frameworks-over-pure-native-JavaScript
18. Should I use a rendering framework or canvas in JavaScript for rendering games? *Noupe - THE magazine for webworkers and site owners* [online]. [cit. 2020-05-01]. Dostupné z: https://www.quora.com/Should-I-use-a-rendering-framework-or-canvas-in-JavaScript-for-rendering-games
19. Should I use an rendering framework or HTML canvas for game development? *Forums - HTML5 Game Devs Forum* [online]. 2019, 7. Mar 2019 [cit. 2020-05-01]. Dostupné z: https://www.html5gamedevs.com/topic/42370-should-i-use-an-rendering-framework-or-html-canvas-for-game-development/

20. Snímková frekvence. *Alza.cz* [online]. [cit. 2020-04-13]. Dostupné z: https://www.alza.cz/snimkova-frekvence
21. Hash: online hash value calculator. *FileFormat.Info* [online]. [cit. 2020-04-13]. Dostupné z: https://www.fileformat.info/tool/hash.htm
22. BĚHÁLEK, Marek. Kódování a hashování. *Katedra informatiky, FEI, VŠB-TU Ostrava* [online]. [cit. 2020-04-13]. Dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch09s01.html
23. Audio and Video – W3C. *World Wide Web Consortium (W3C)* [online]. [cit. 2020-05-08]. Dostupné z: https://www.w3.org/standards/webdesign/audiovideo
24. Unity WebGL Player: unity_bench2. *Unity WebGL Player* [online]. 2015 [cit. 2020-04-13]. Dostupné z: https://files.unity3d.com/jonas/benchmark2015/
25. Introduction to polygons. *Autodesk Knowledge Network* [online]. 2015, 23 Jan 2015 [cit. 2020-04-13]. Dostupné z: https://knowledge.autodesk.com/support/maya-lt/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/MayaLT/files/Polygons-overview-Introduction-to-polygons-htm.html
26. Fast and Simple JavaScript FPS Counter. *Growing with the Web* [online]. 2017, 30 Dec 2017 [cit. 2020-04-13]. Dostupné z: https://www.growingwiththeweb.com/2017/12/fast-simple-js-fps-counter.html
27. SubtleCrypto.digest(). *MDN web docs* [online]. 2020, 23 Mar 2020 [cit. 2020-04-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/digest
28. Freesound - "Coins 2" by DominikBraun. *Freesound - Freesound* [online]. [cit. 2020-05-02]. Dostupné z: https://freesound.org/people/DominikBraun/sounds/483507/
29. Convert Audio Files To/From All Audio Formats with Switch. *CH Software - Free Software Downloads and Installs* [online]. [cit. 2020-05-02]. Dostupné z: https://www.nch.com.au/switch/index.html
30. PixiJS Sound | Basics. *PixiJS* [online]. [cit. 2020-05-02]. Dostupné z: https://pixijs.io/pixi-sound/examples/index.html
31. Male Base Mesh Free 3D Model - .obj - Free3D. *3D Models for Free - Free3D.com* [online]. [cit. 2020-05-02]. Dostupné z: https://free3d.com/3d-model/male-base-mesh-6682.html
32. Load from any file type – glTF, OBJ, STL, etc. *Babylon.js Documentation* [online]. [cit. 2020-05-02]. Dostupné z: https://doc.babylonjs.com/how_to/load_from_any_file_type
33. Lights. *Babylon.js Documentation* [online]. [cit. 2020-05-02]. Dostupné z: https://doc.babylonjs.com/babylon101/lights
34. Apatche Cordova. *Apatche Cordova* [online]. [cit. 2020-05-02]. Dostupné z: https://cordova.apache.org/
35. The config.xml File – Apache Cordova. *Apache Cordova* [online]. [cit. 2020-05-08]. Dostupné z: https://cordova.apache.org/docs/en/3.1.0/config_ref/index.html
36. Cordova file plugin – save file in device. *Stack Overflow* [online]. [cit. 2020-05-02]. Dostupné z: https://stackoverflow.com/questions/38832592/cordova-file-plugin-save-file-in-device
37. File – Apache Cordova. *Apatche Cordova* [online]. [cit. 2020-05-02]. Dostupné z: https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-file/#write-to-a-file-

Seznam příloh

V přiloženém ZIP archívu naleznete následující adresáře:

- **/Výsledky** – seznam tabulek s detailním vyhodnocením výsledků
- **/pixi** – řešení implementace testů pomocí frameworku Pixi.js
- **/phaser** – řešení implementace testů pomocí frameworku Phaser
- **/babylon** – řešení implementace testů pomocí frameworku Babylon
- **/apk** – instalační balíčky aplikací pro OS Android
- **/ApacheCordova_projekty** – Apache Cordova projekty, užívané k exportu do apk
- **/readme.txt** – textový soubor s návodem pro spuštění projektů ve webovém prohlížeči